Introduction to Artificial Intelligence

丁尧相
浙江大学

Fall & Winter 2022
Week 13

Announcements

- Problem set 4 (last homework) will be released this week.
- Lab project 1 is due today.
- We will release lab project 3 this week. Both Project 2 & 3 are due on Dec. 31.
- You can choose 2 out of 3 lab projects to get all normal scores.
- You can get bonus scores if you finish all 3 lab projects.

Machine Learning: IV

- Convolutional network
- Deep learning
  - Gradient vanishing
  - Deep NN architectures
  - Tricks of the trade
- Take-home messages

Logistic Regression

- Logistic regression (LR) is a binary classification model.
- LR builds upon the generalized linear model to represent $p(y|\mathbf{x})$:

$$p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

- $p(x) = \frac{1}{1+e^{-x}}$ is called sigmoid function, which is monotonically increasing.

Larger $\mathbf{w}^T\mathbf{x}$ leads to larger $p(y|\mathbf{x})$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \min_{\theta} \sum_{i=1}^{N} \left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^{N} \underbrace{\left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]}_{\text{logistic loss}}$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^{N} \underbrace{\left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]}_{\text{logistic loss}} \qquad \underbrace{\max\left(0, 1-y_i(\mathbf{w}^T\mathbf{x}_i+\mathbf{b})\right)}_{\text{hinge loss}}$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^{N} \left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]$$

logistic loss

$$\max\left(0, 1-y_i(\mathbf{w}^T\mathbf{x}_i+\mathbf{b})\right)$$

hinge loss

Logistic regression can also be formulated into loss minimization.
Regularization and kernel method can also be used!

Cross-Entropy Loss

- Logistic loss $\log(1+e^{-y\mathbf{w}^T\mathbf{x}})$ is designed for linear model in binary classification.

- For general probabilistic model $f(\mathbf{x})$, we can design a similar loss:

$$-\Big[\mathbb{I}[y=+1]log(p)+\mathbb{I}[y=-1]log(1-p)\Big]$$

p is the abbreviation of $p(x)$

This is called the cross-entropy loss for binary classification.
It is equivalent to logistic regression when using linear model.

Multi-Class Cross-Entropy Loss

- For multi-class classification with $y \in \{1, 2, ..., C\}$, extend logistic regression model as

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \quad \rightarrow \quad p(y_c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_{i=1}^{C} e^{\mathbf{w}_c^T\mathbf{x}_i}}$$

There is a parameter $\mathbf{w}_c$ for each class.

- The multi-class logistic loss is

$$\log(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \quad \rightarrow \quad -\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

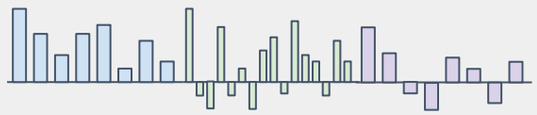
- In general, $p(y_c|\mathbf{x})$ can represented by other functions, then the multi-class cross-entropy loss is also

$$-\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

Usually the most common choice for classification with NN.

From Linear Model to Neural Network

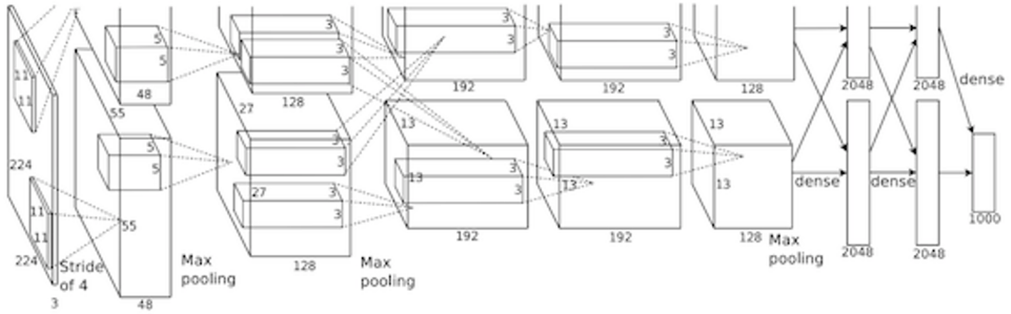
Feature Extraction



f

training

10 numbers giving scores for classes



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

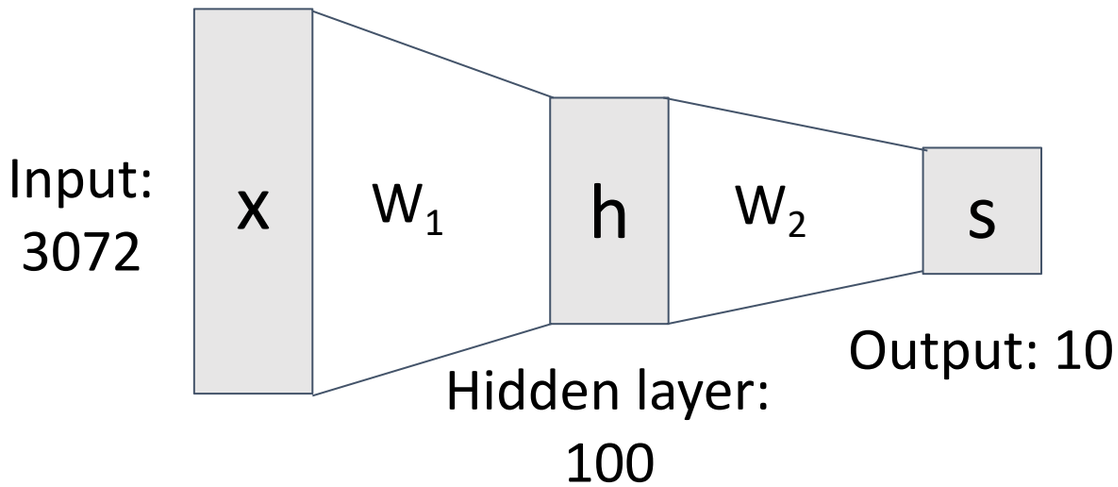
training

10 numbers giving scores for classes

Neural networks have more complicated function mapping.
Handle complex data. Need less feature extraction.

Feedforward Neural Network

All elements of x affect all elements of h

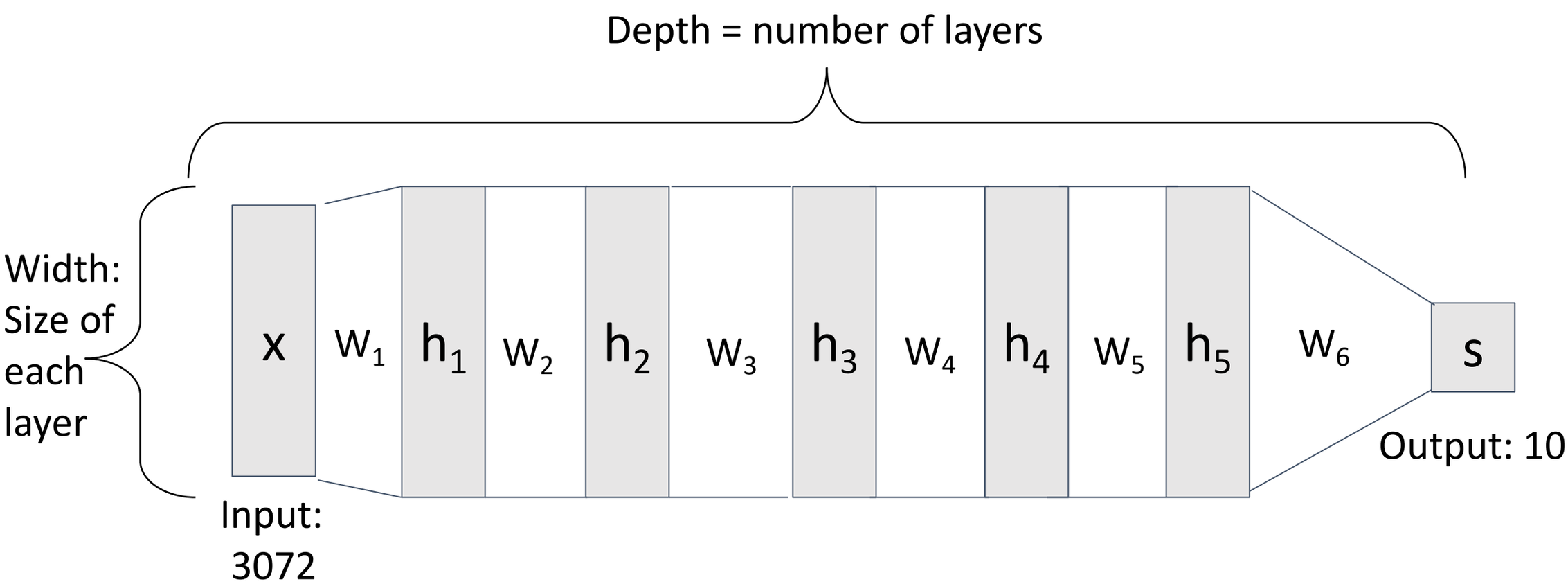


All elements of h affect all elements of s

2-layer Neural Network $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

Called Fully-connected feedforward neural network.

Neural Network with More Layers



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Fully-connected NNs with multiple hidden layers are usually called multi-layer perceptron (MLP).

SGD = Stochastic + Gradient

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$

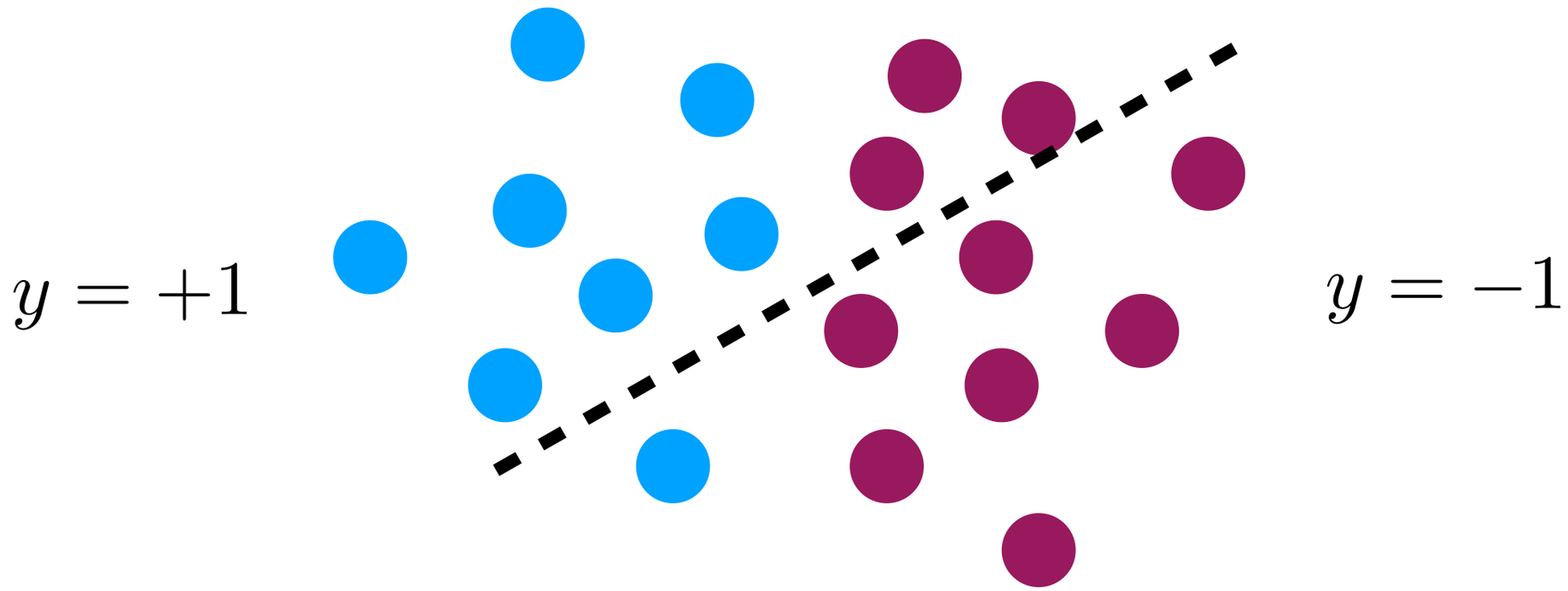
- Gradient descent uses all training data to calculate the loss.
- In each iteration, stochastic gradient descent (SGD) randomly samples a small subset of data to calculate the loss.

More computational efficient.
Lead to better solutions for NNs.

The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

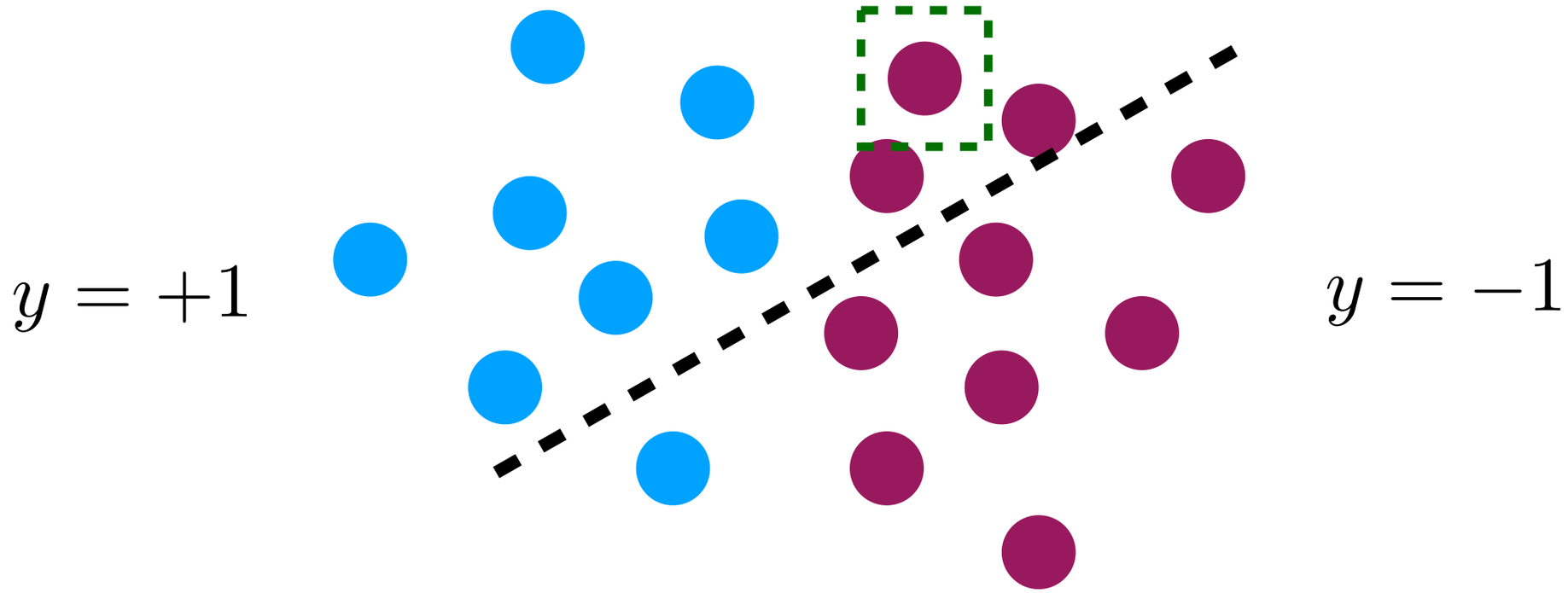
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

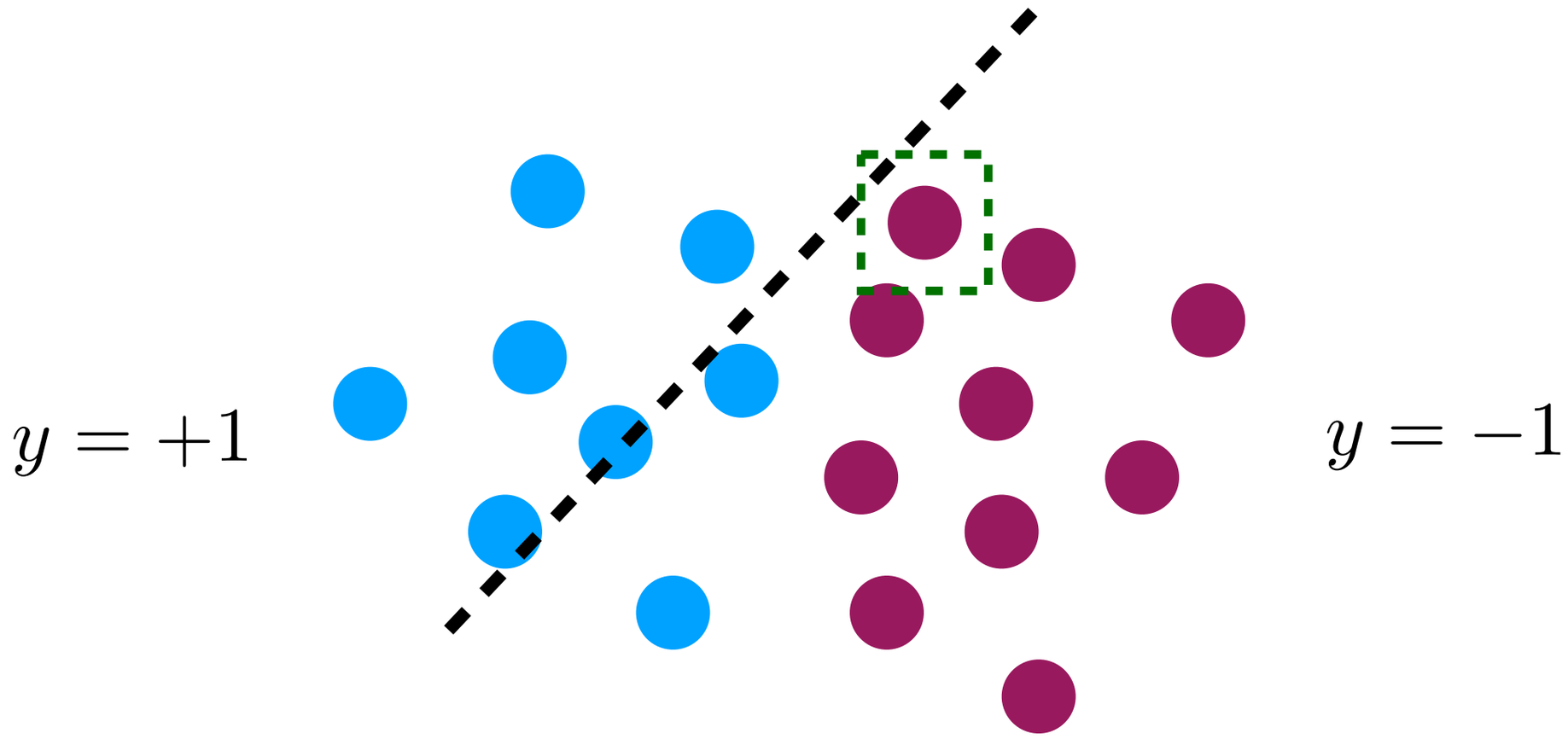
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

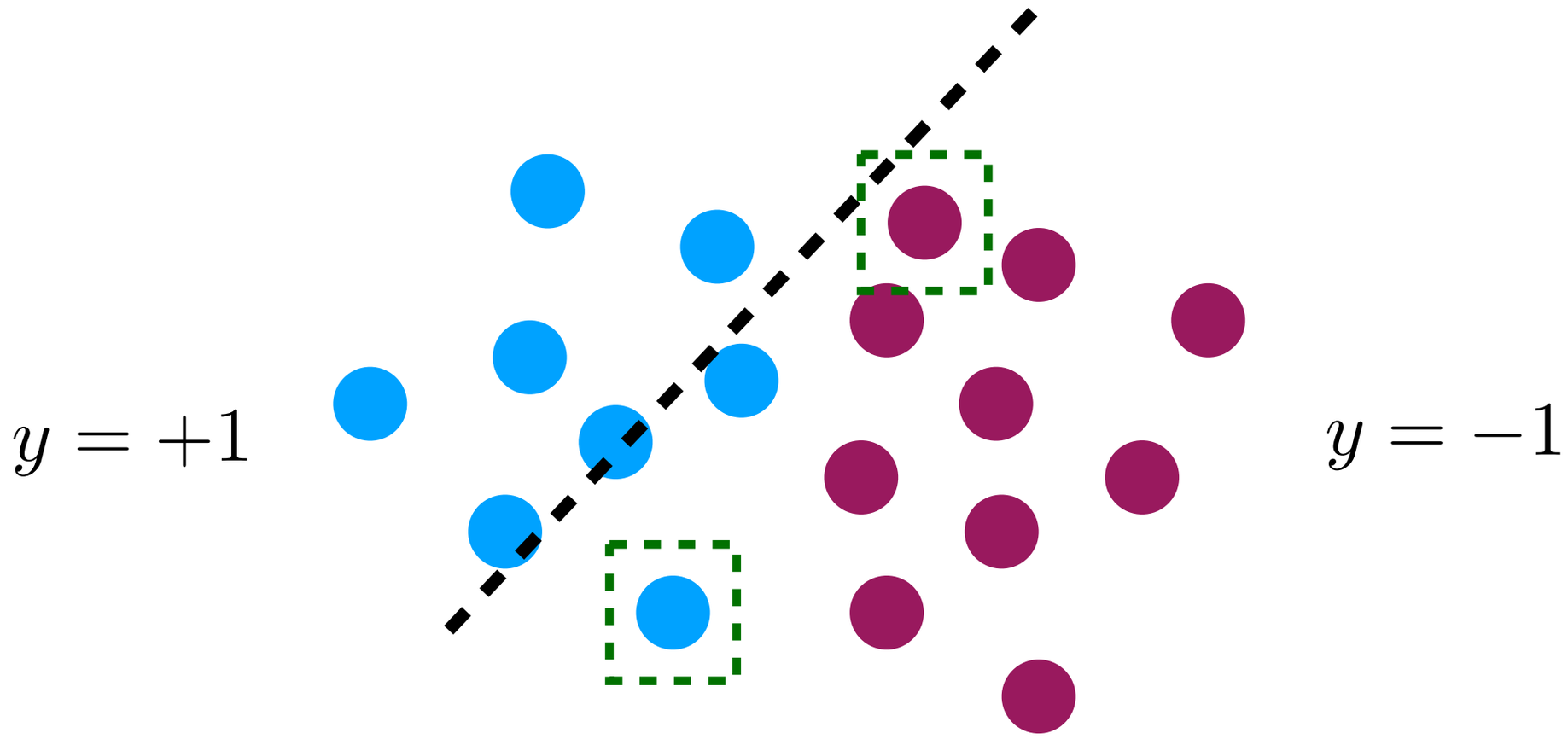
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

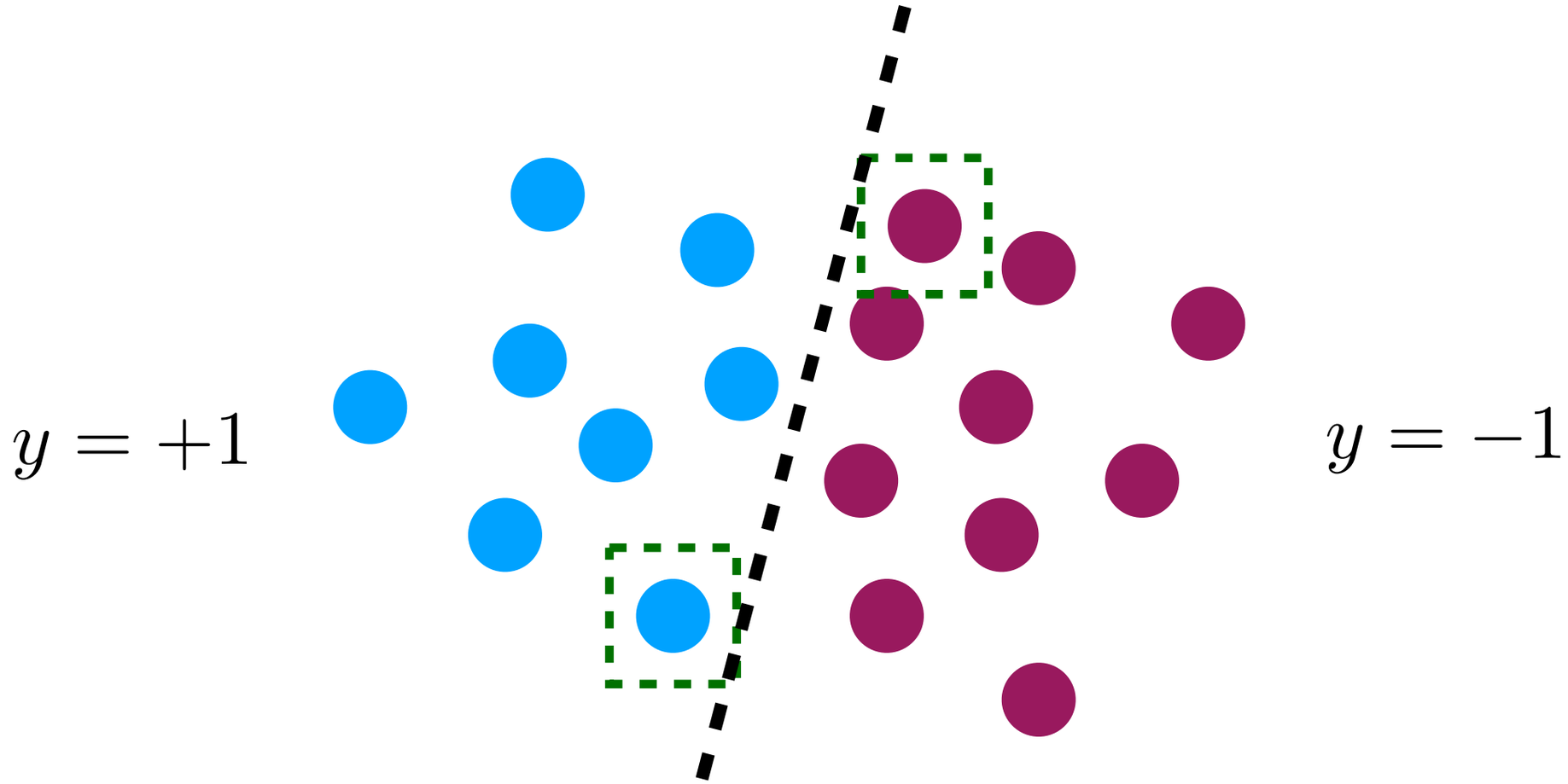
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

Random sample instances and fix the hyperplane according to them.



SGD for Multi-Layer NN

- NNs are trained under various of objective functions. For regression, the common choice is MSE. For classification, the cross-entropy loss is often used:

$$-\sum_{i=1}^{N'}\Big[\sum_{c=1}^{C}\mathbb{I}[y_i = f(\mathbf{x}_i)]\log f(\mathbf{x}_i)\Big]$$

SGD: sample a small subset of data to calculate

Given an initial weight vector $\boldsymbol{w}_0$,

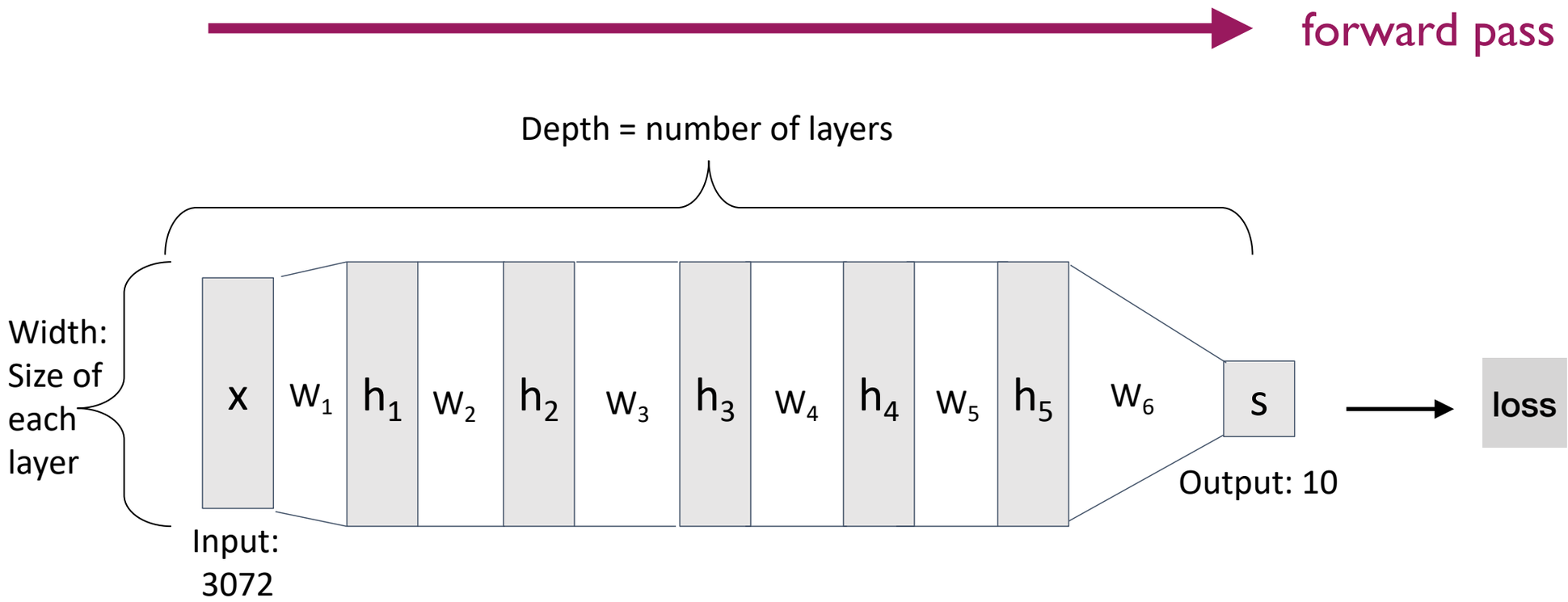
Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$

Parameter of NNs are the weights for all layers.
The key is to obtain their gradients efficiently.

Back Propagation



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

backward pass

NNs can be formulated into computational graphs, in which nodes represent values and parameters, edges represent operators.
Both training and prediction can be realized by forward and backward passes through the computational graphs.

Machine Learning: IV

- Convolutional network

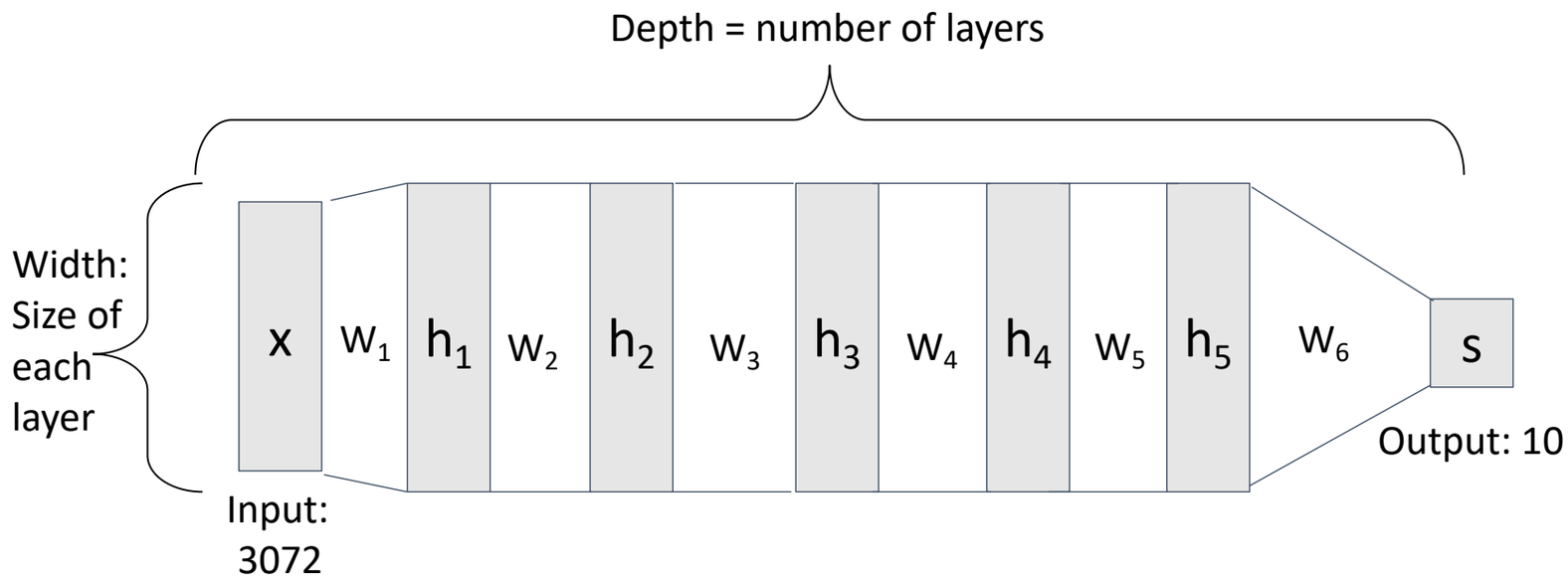
- Deep learning
  - Gradient vanishing and representation learning
  - Deep NN architectures
  - Tricks of the trade

- Take-home messages

Model Complexity of Fully-Connected NN

- Assume that all layers have d units, if the networks has depth b, then the number of weight parameters are bd^2.

- For fully-connected NN, we usually assume $d = 128 \sim 1024$.



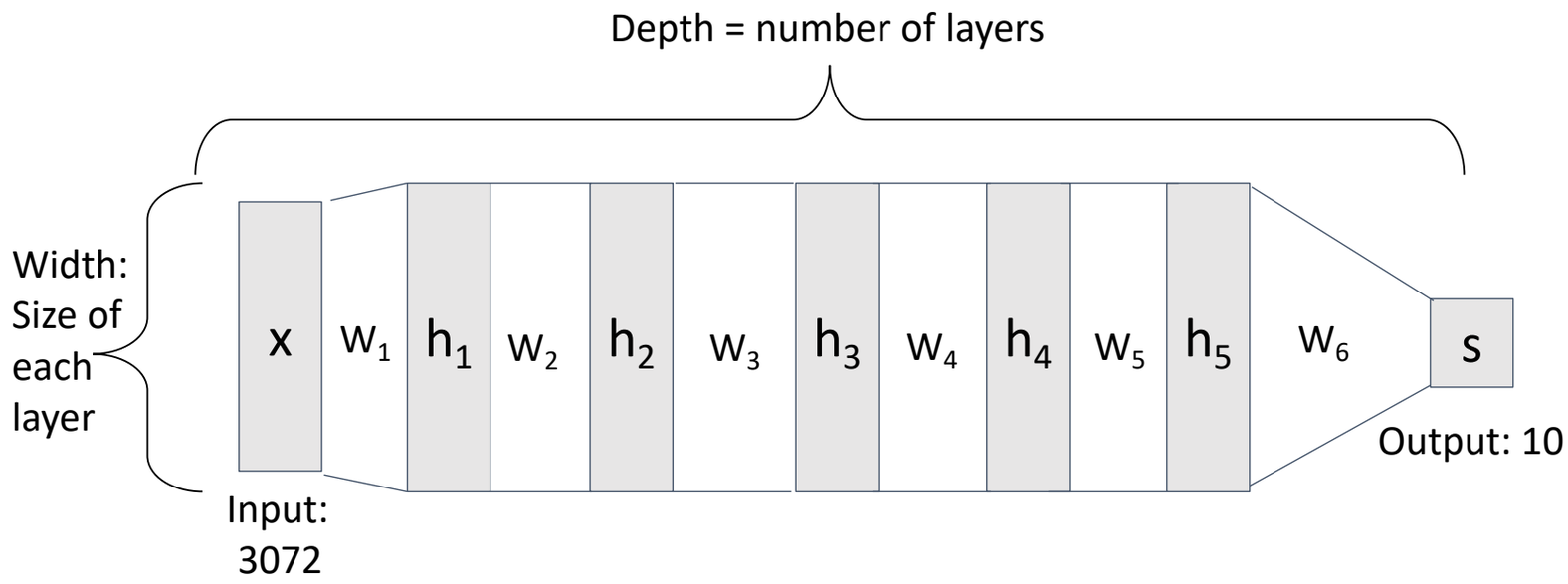
$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Model Complexity of Fully-Connected NN

- Assume that all layers have d units, if the networks has depth b, then the number of weight parameters are bd^2.

- For fully-connected NN, we usually assume $d = 128 \sim 1024$.

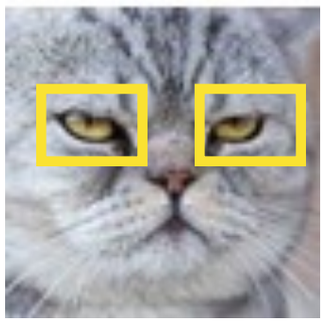
If $b = 10, d = 512$, the number of weight parameters is over 2 million.
We need a systematic way to reduce the number of parameters.



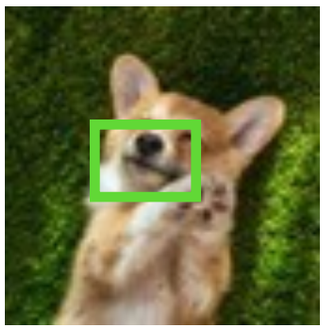
$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Local Structures in Data

- For common data such as images and natural languages, there usually exist significant local structures.

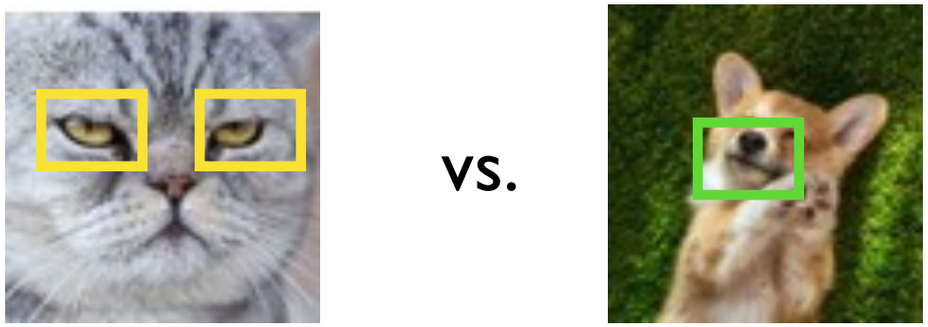


vs.



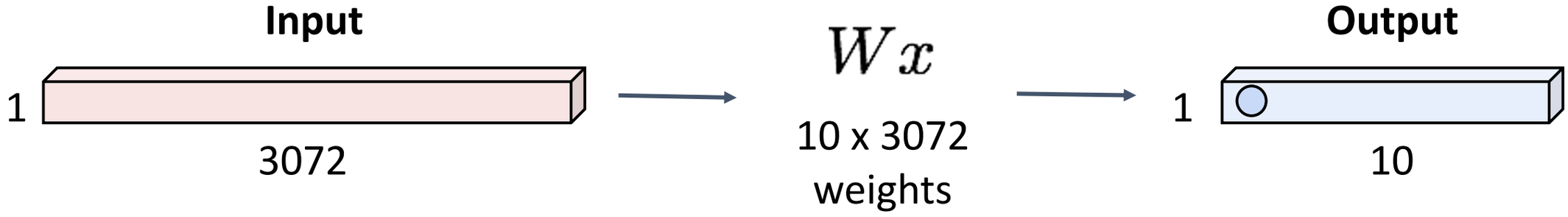
Local Structures in Data

- For common data such as images and natural languages, there usually exist significant local structures.



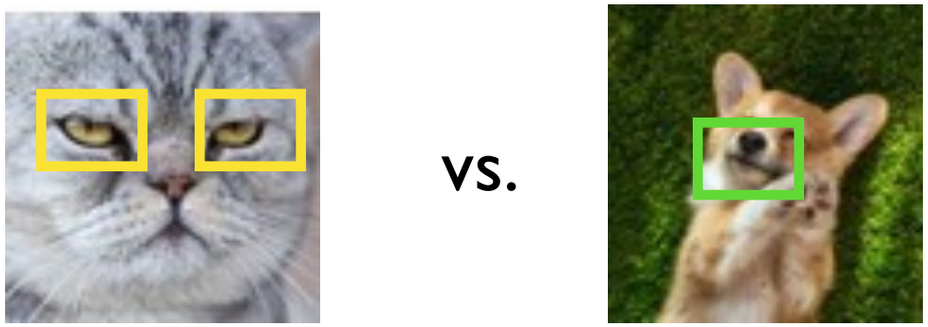
Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1



Local Structures in Data

- For common data such as images and natural languages, there usually exist significant local structures.

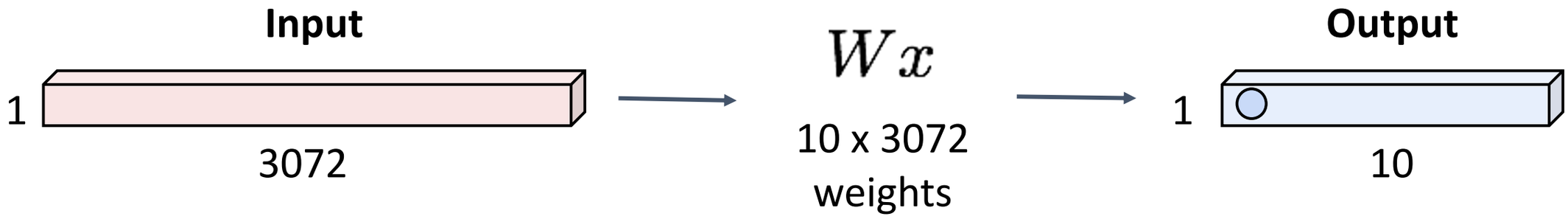


vs.

Fully-Connected Layer

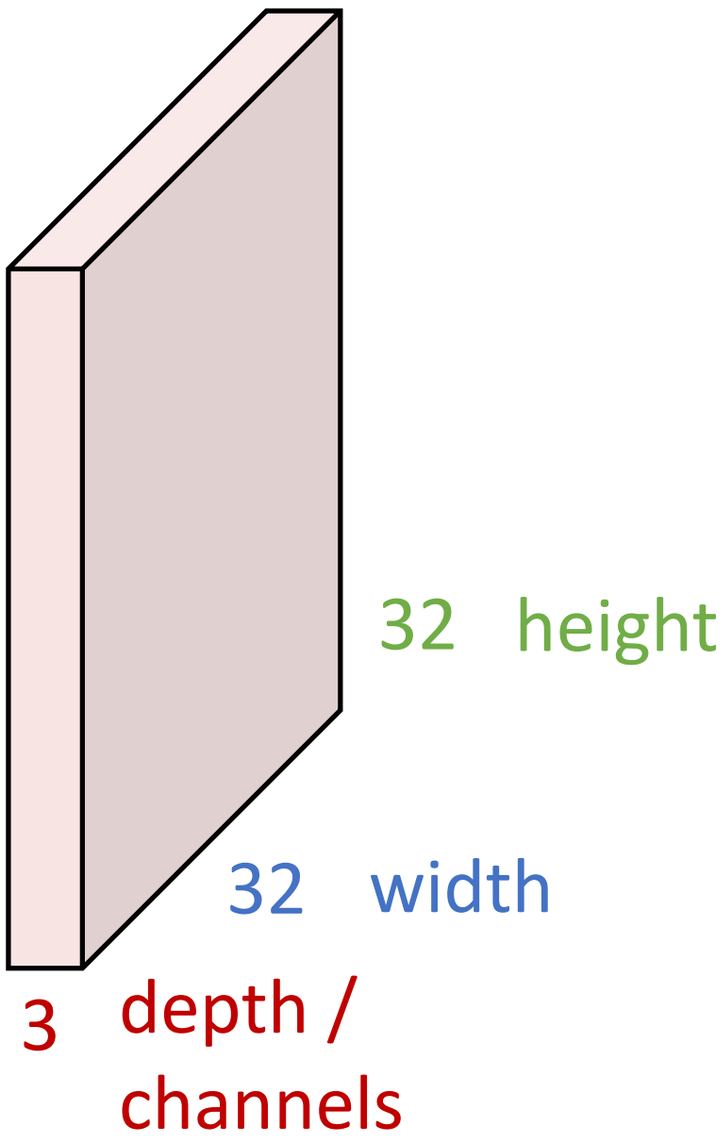
32x32x3 image -> stretch to 3072 x 1

One unit in the next layer is connected to all units in the previous layer.
Can't represent local structure!



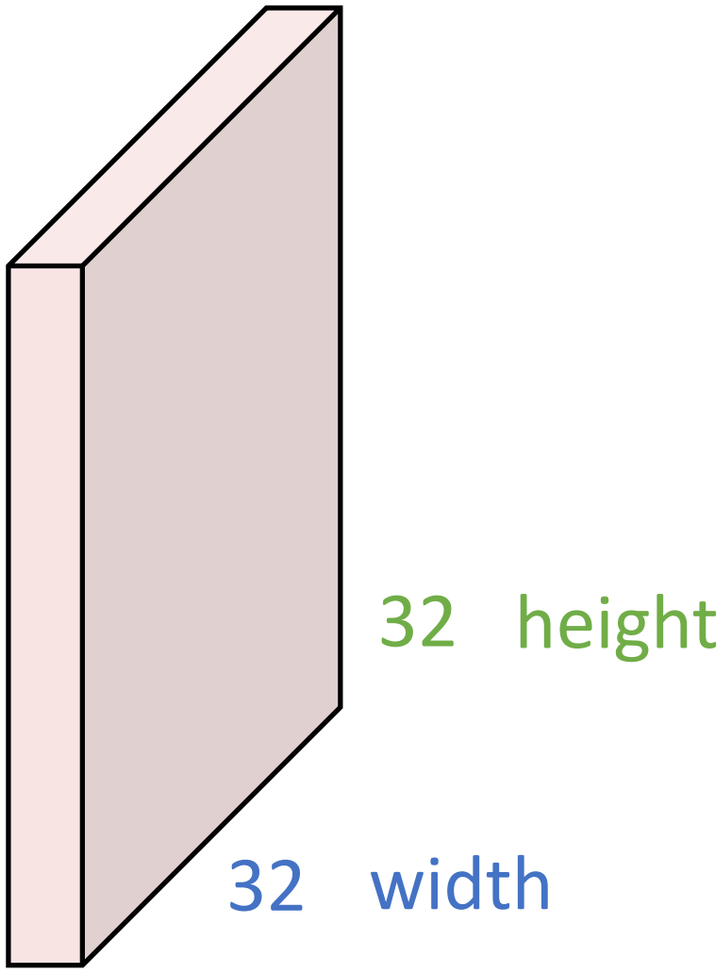
Convolution Layer

3x32x32 image

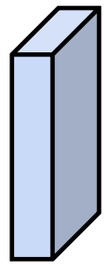


Convolution Layer

3x32x32 image



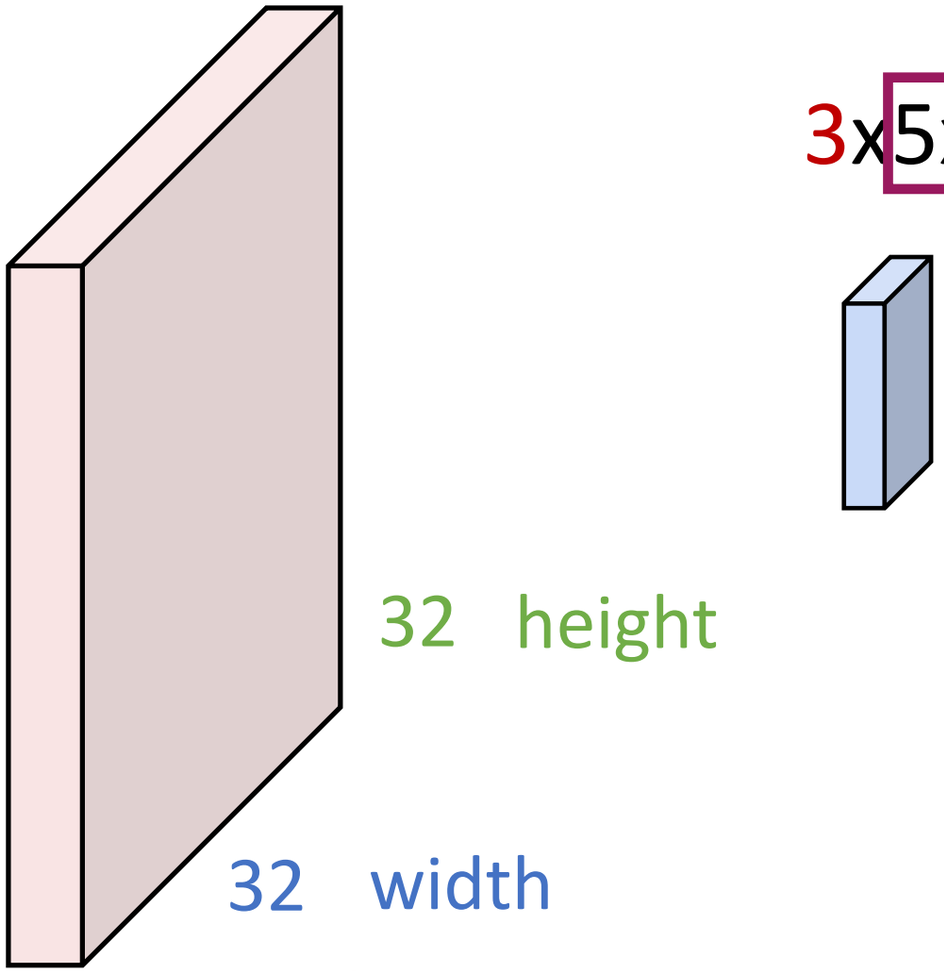
3x5x5 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolution Layer

3x32x32 image

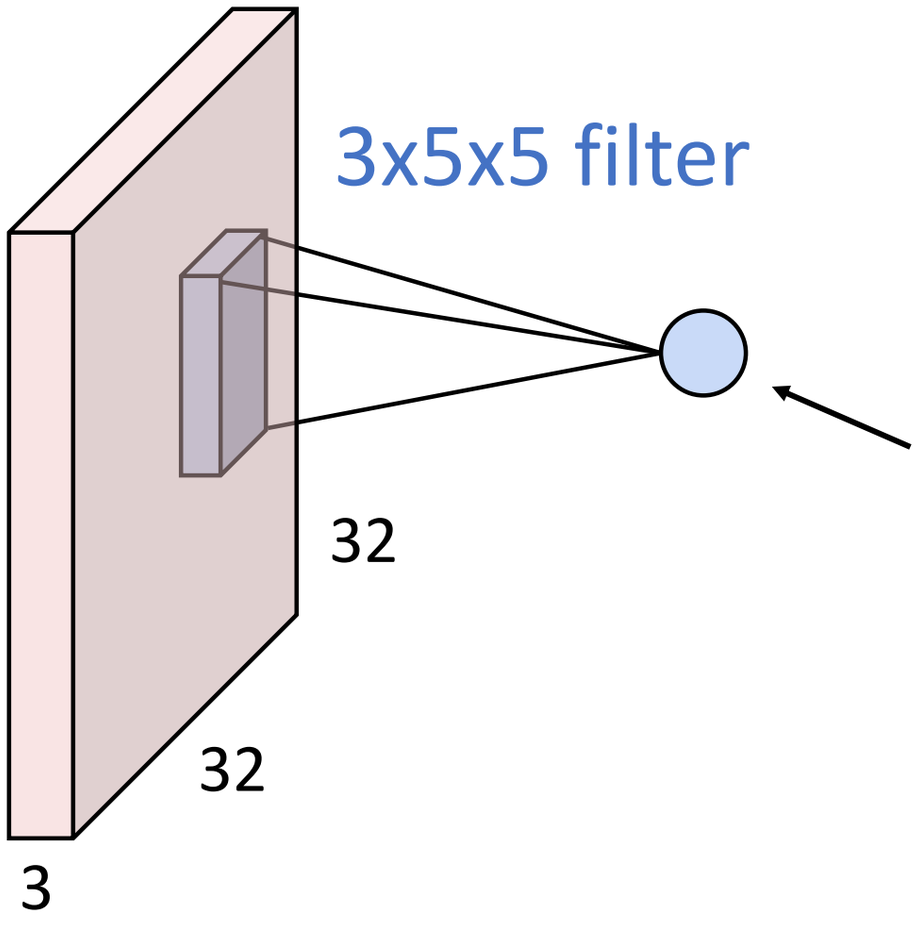


where the local structure lies on

3x5x5 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

For simplicity, we only show $3 \times 2 \times 2$ filter

x

0.4	0.2
-0.3	-0.5

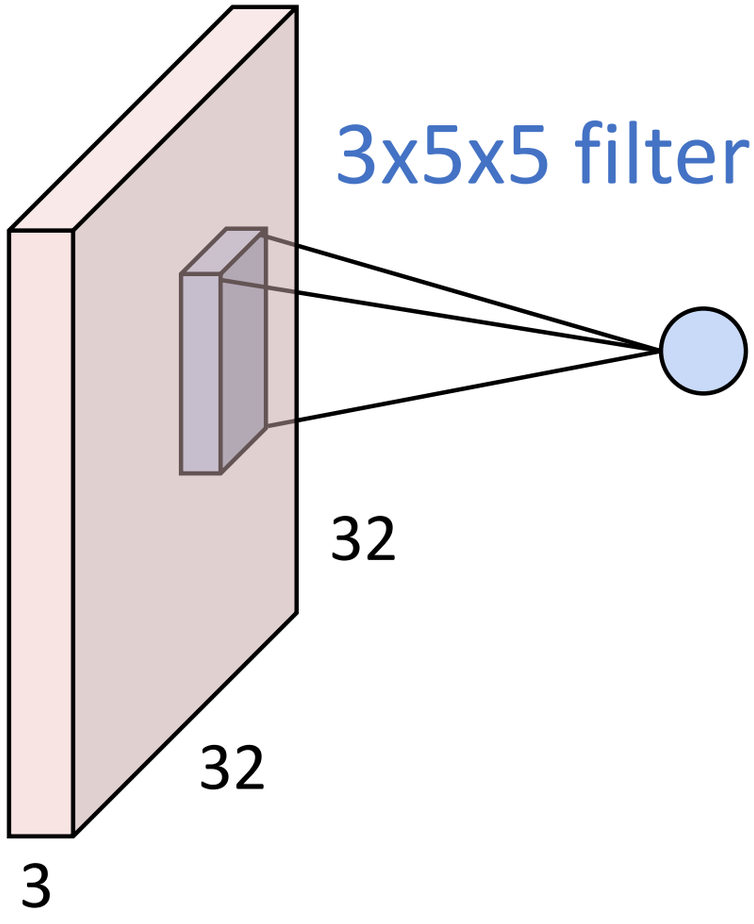
-0.3	-0.2
0.5	0.6

w

0.7	-0.2
0.8	-0.3

0.9	-0.5
0.3	0.4

3x32x32 image



-0.4	0.3
-0.8	0.5

0.8	0.7
-0.5	0.6

1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

For simplicity, we only show $3 \times 2 \times 2$ filter

Dot products are conducted by stretching matrix into vector

x

0.4	0.2
-0.3	-0.5

dot product

w

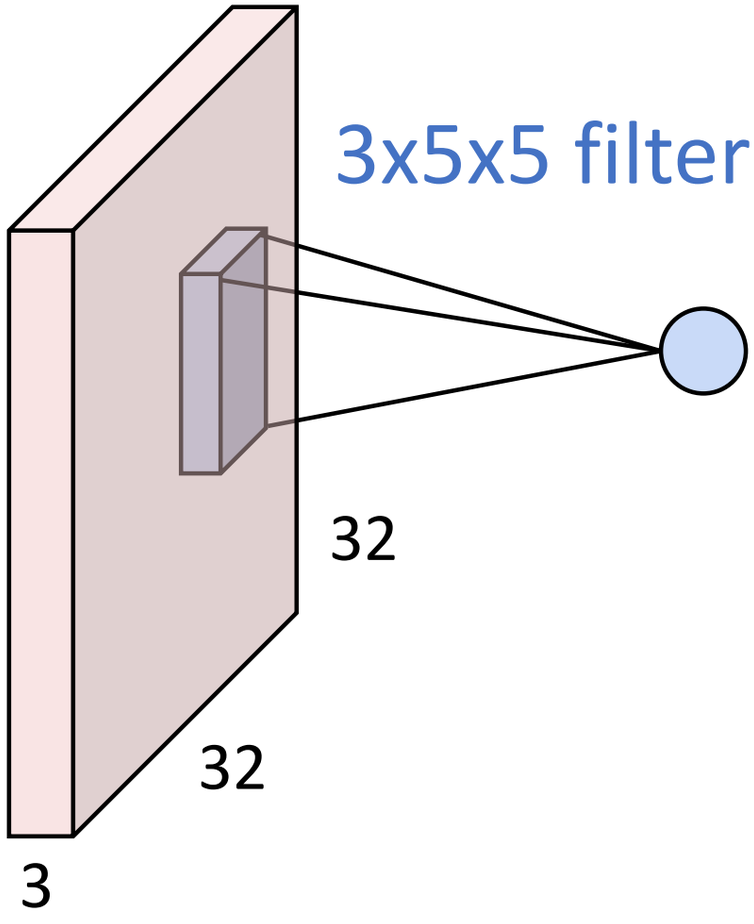
0.7	-0.2
0.8	-0.3

-0.3	-0.2
0.5	0.6

dot product

0.9	-0.5
0.3	0.4

3x32x32 image



-0.4	0.3
-0.8	0.5

dot product

0.8	0.7
-0.5	0.6

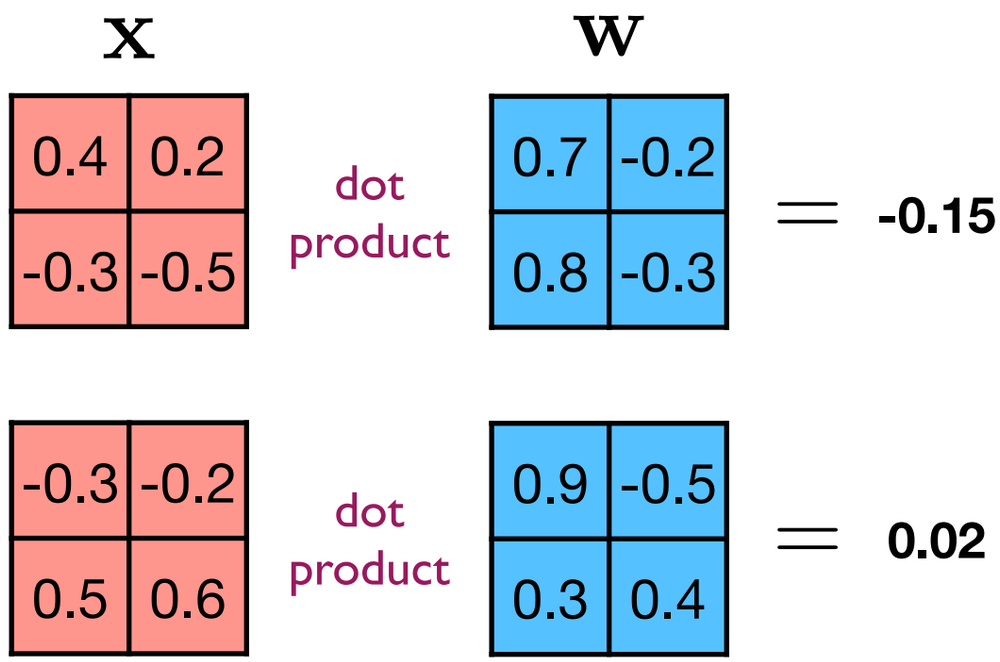
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

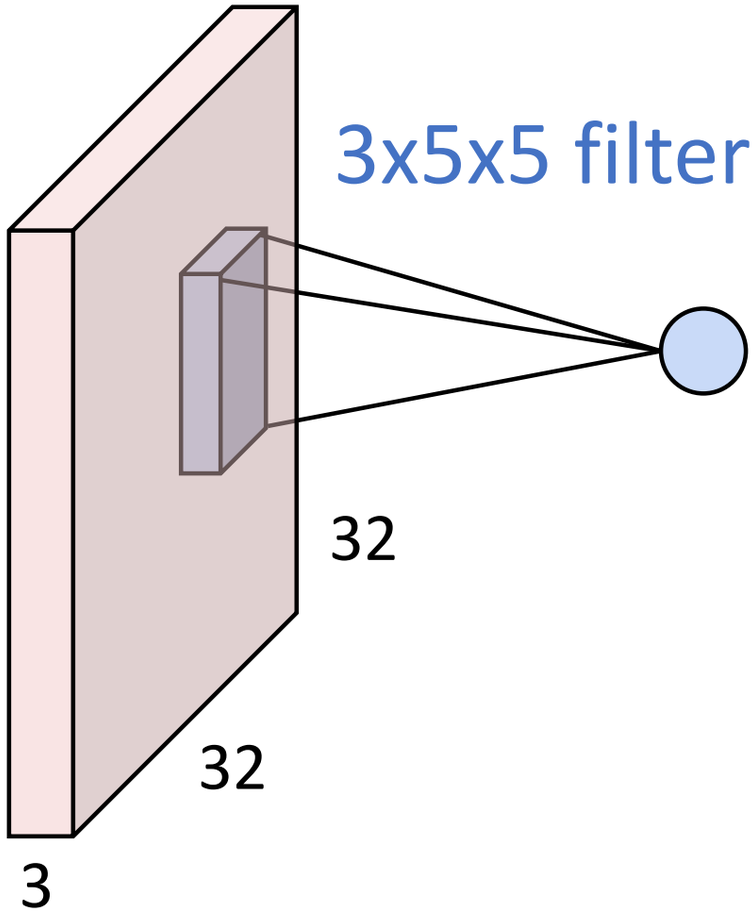
Convolution Layer

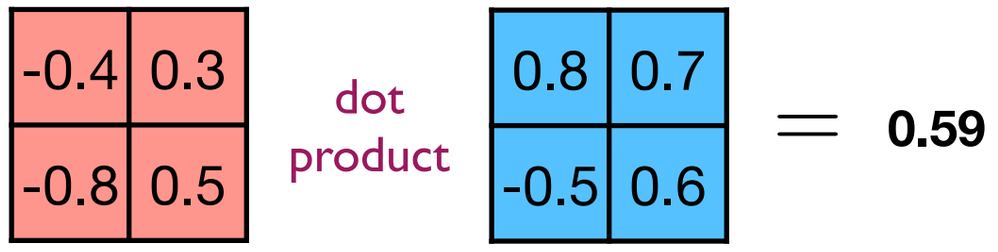
For simplicity, we only show $3 \times 2 \times 2$ filter

Dot products are conducted by stretching matrix into vector



3x32x32 image





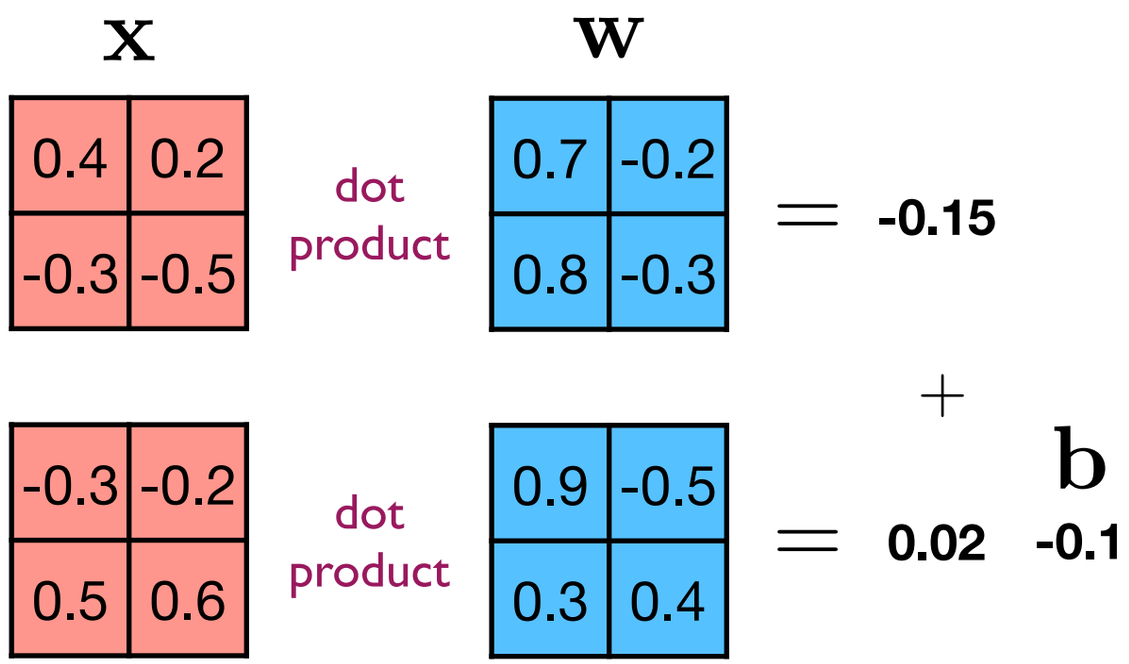
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

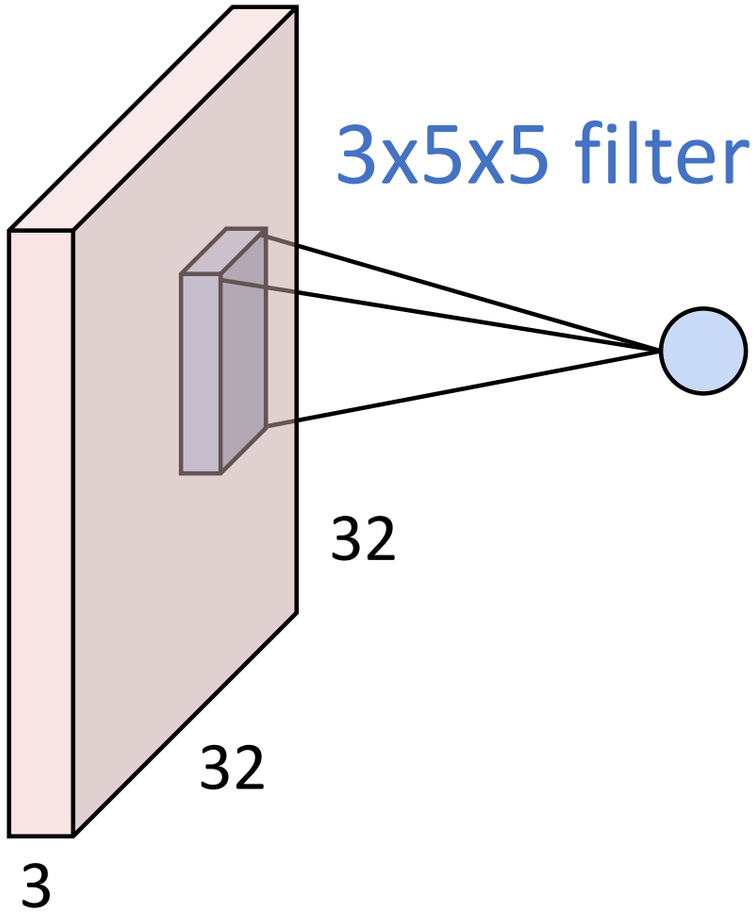
Convolution Layer

For simplicity, we only show $3 \times 2 \times 2$ filter

Dot products are conducted by stretching matrix into vector



3x32x32 image



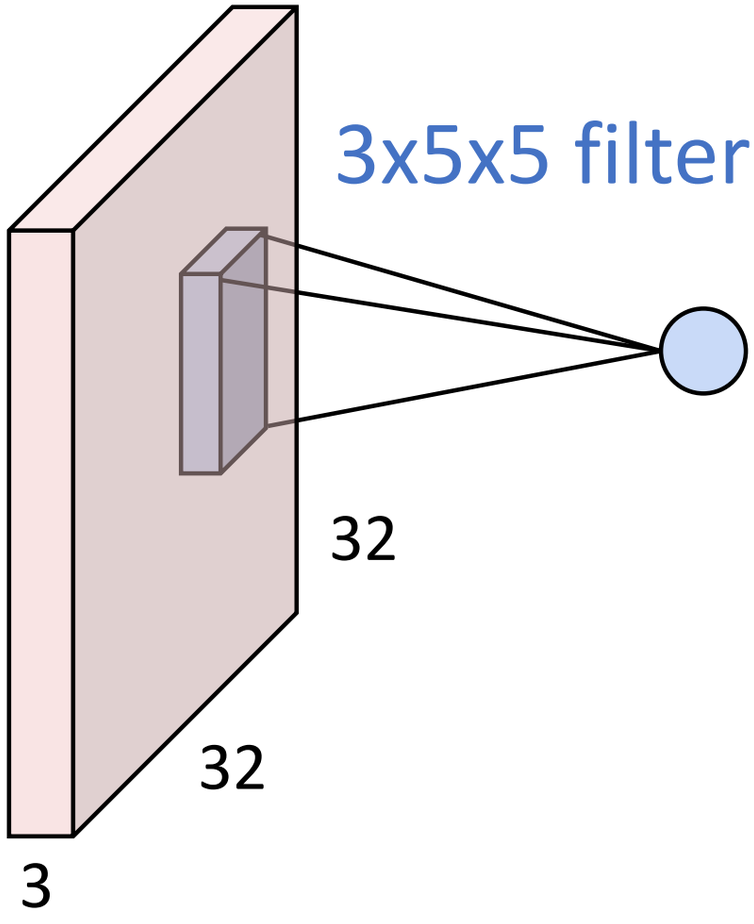
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

For simplicity, we only show $3 \times 2 \times 2$ filter

3x32x32 image



x		w	
0.4	0.2	0.7	-0.2
-0.3	-0.5	0.8	-0.3

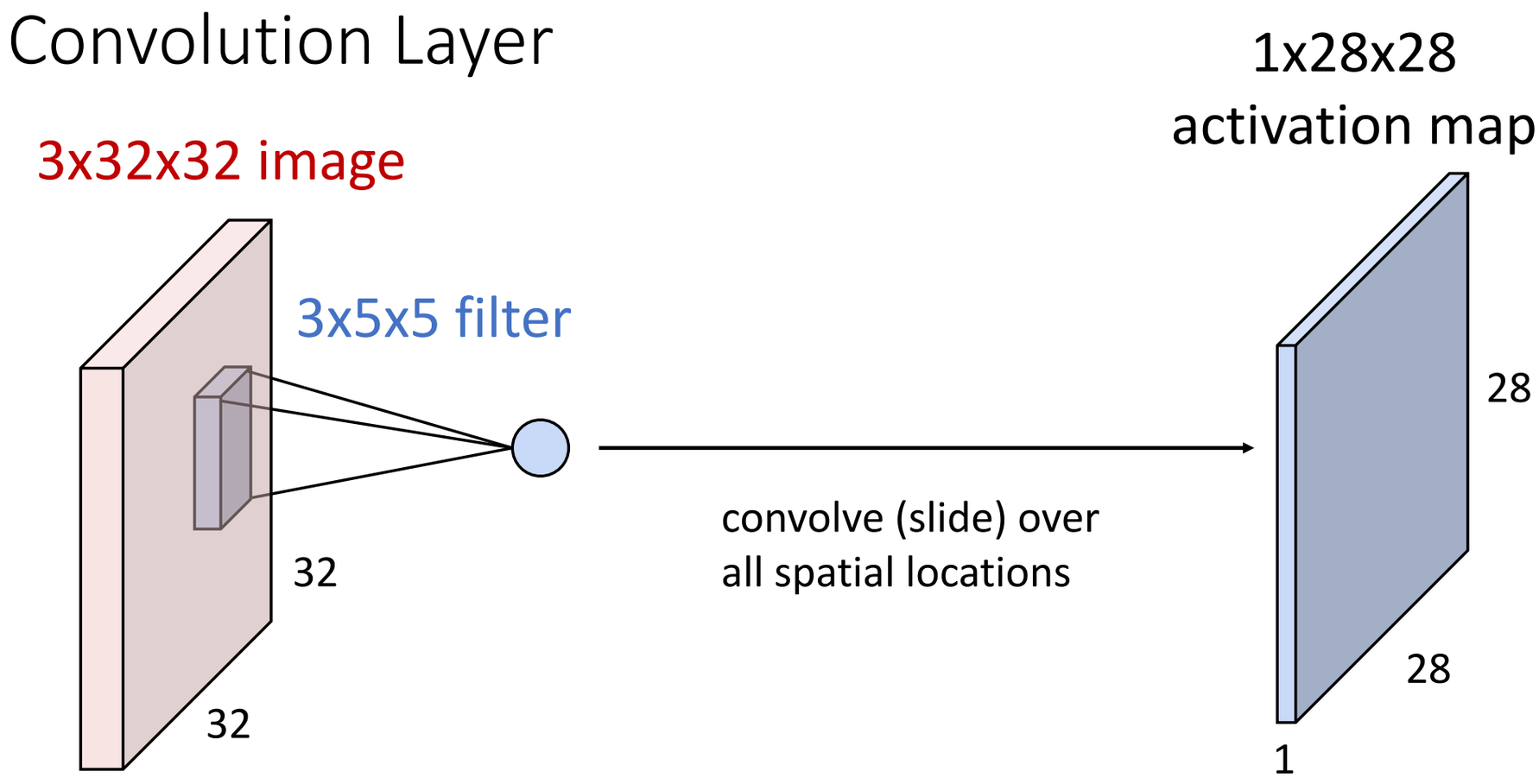
-0.3	-0.2	conv	0.9	-0.5	= 0.36
0.5	0.6		0.3	0.4	

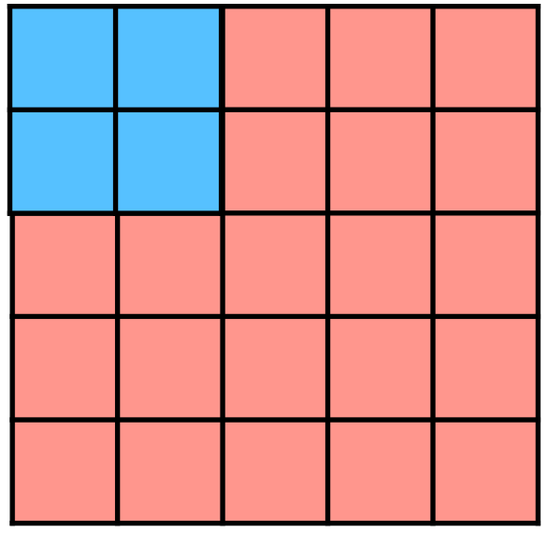
-0.4	0.3	0.8	0.7
-0.8	0.5	-0.5	0.6

1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

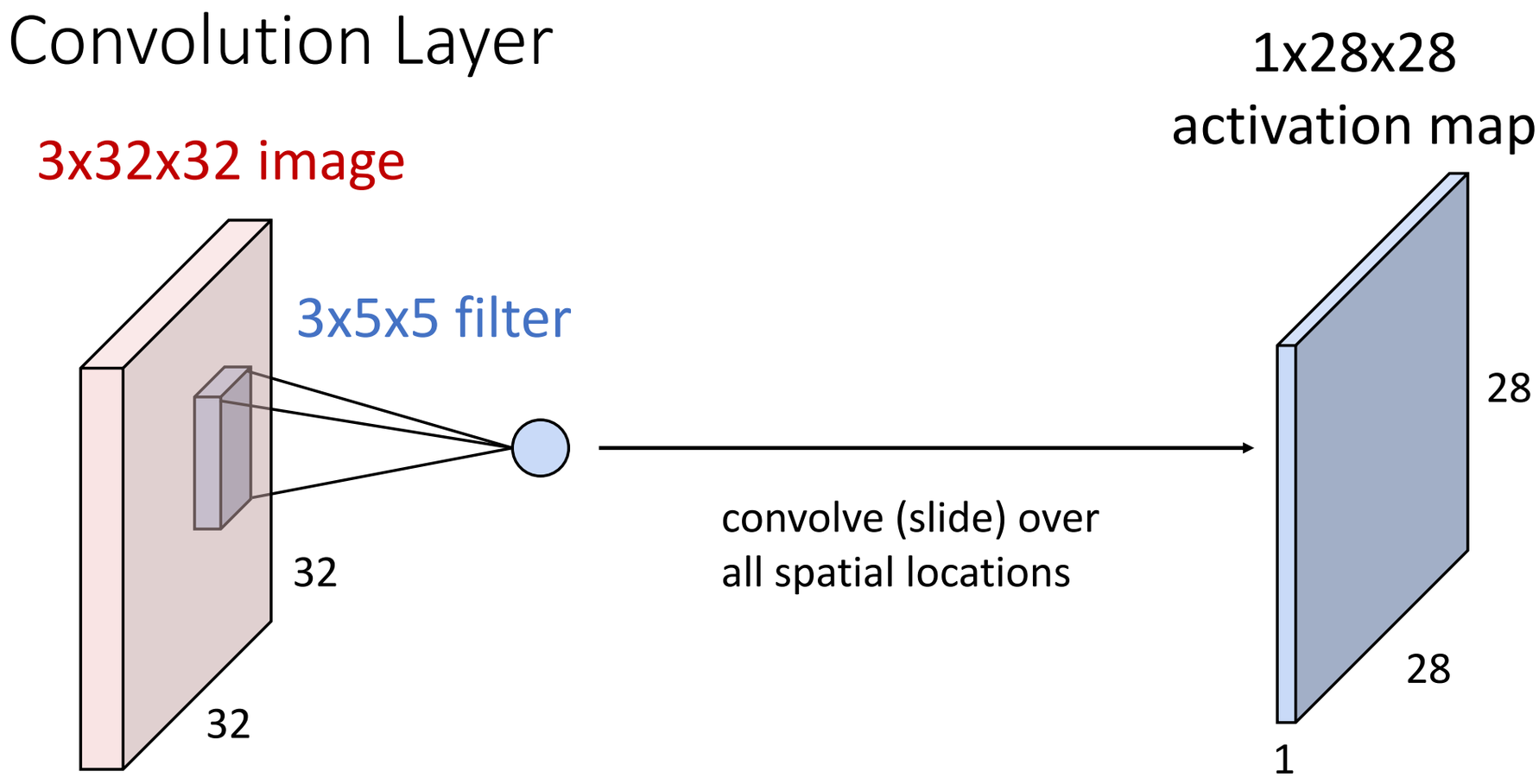
$$w^T x + b$$

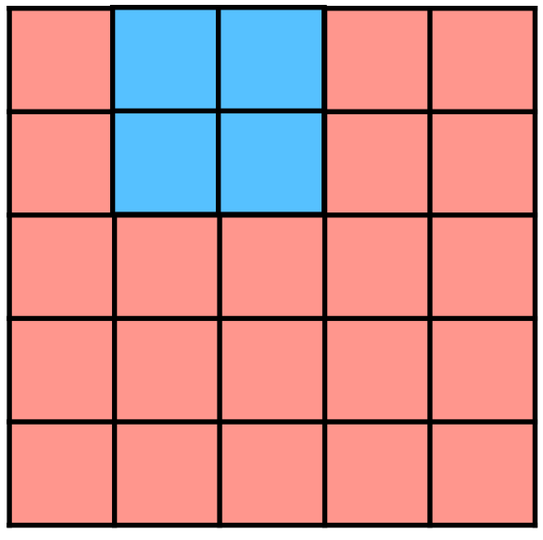
Convolution Layer



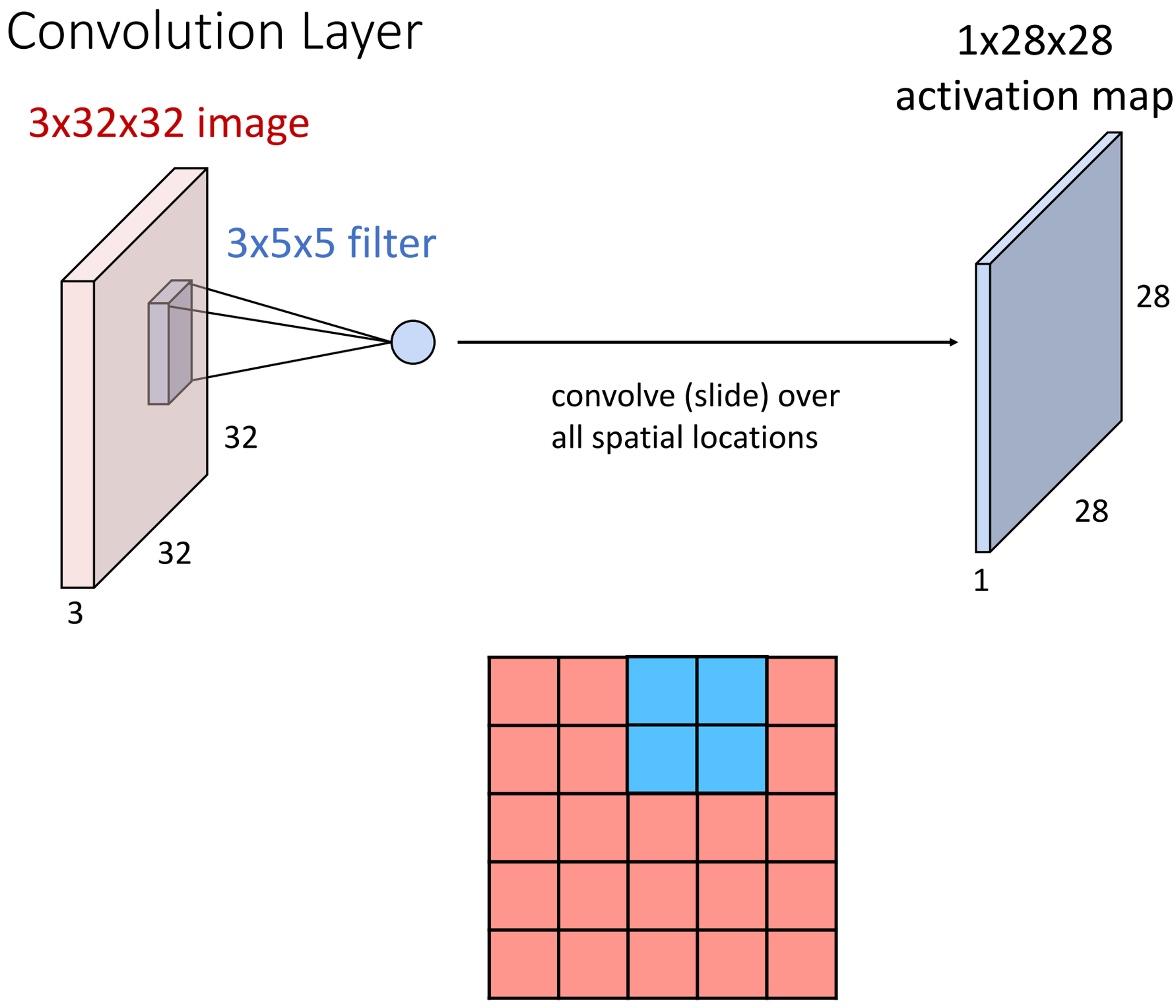


Convolution Layer

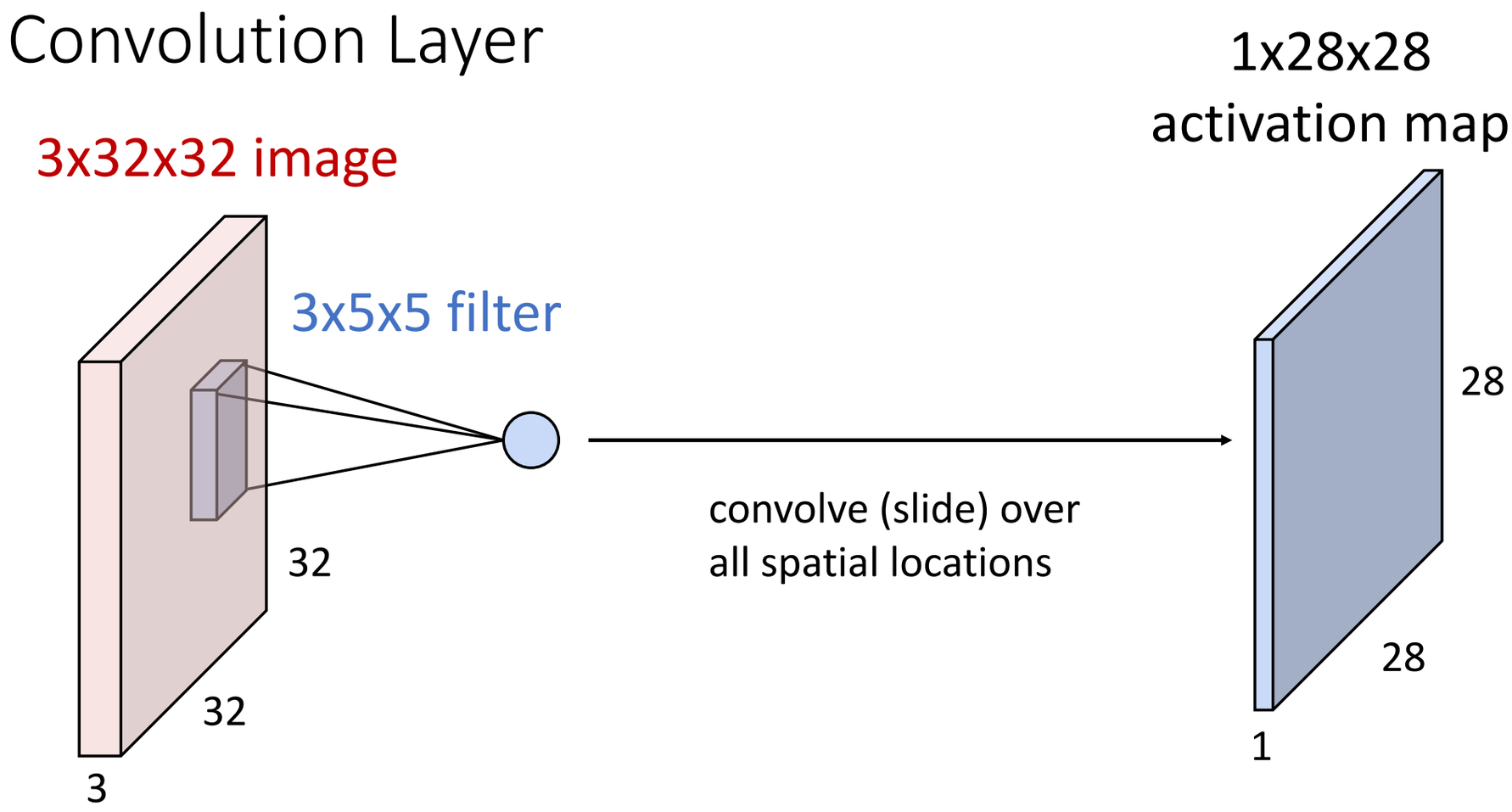


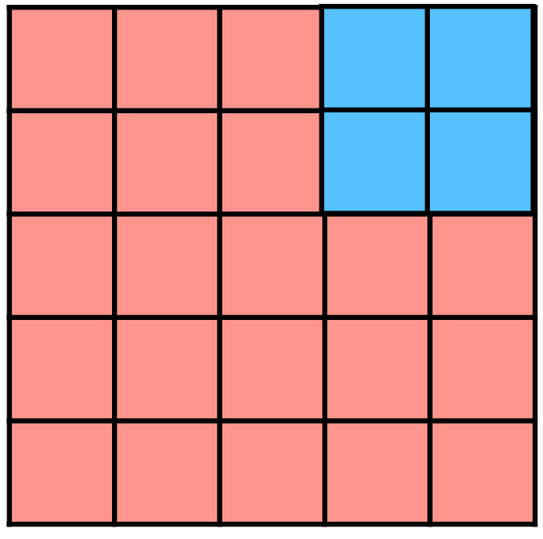


Convolution Layer

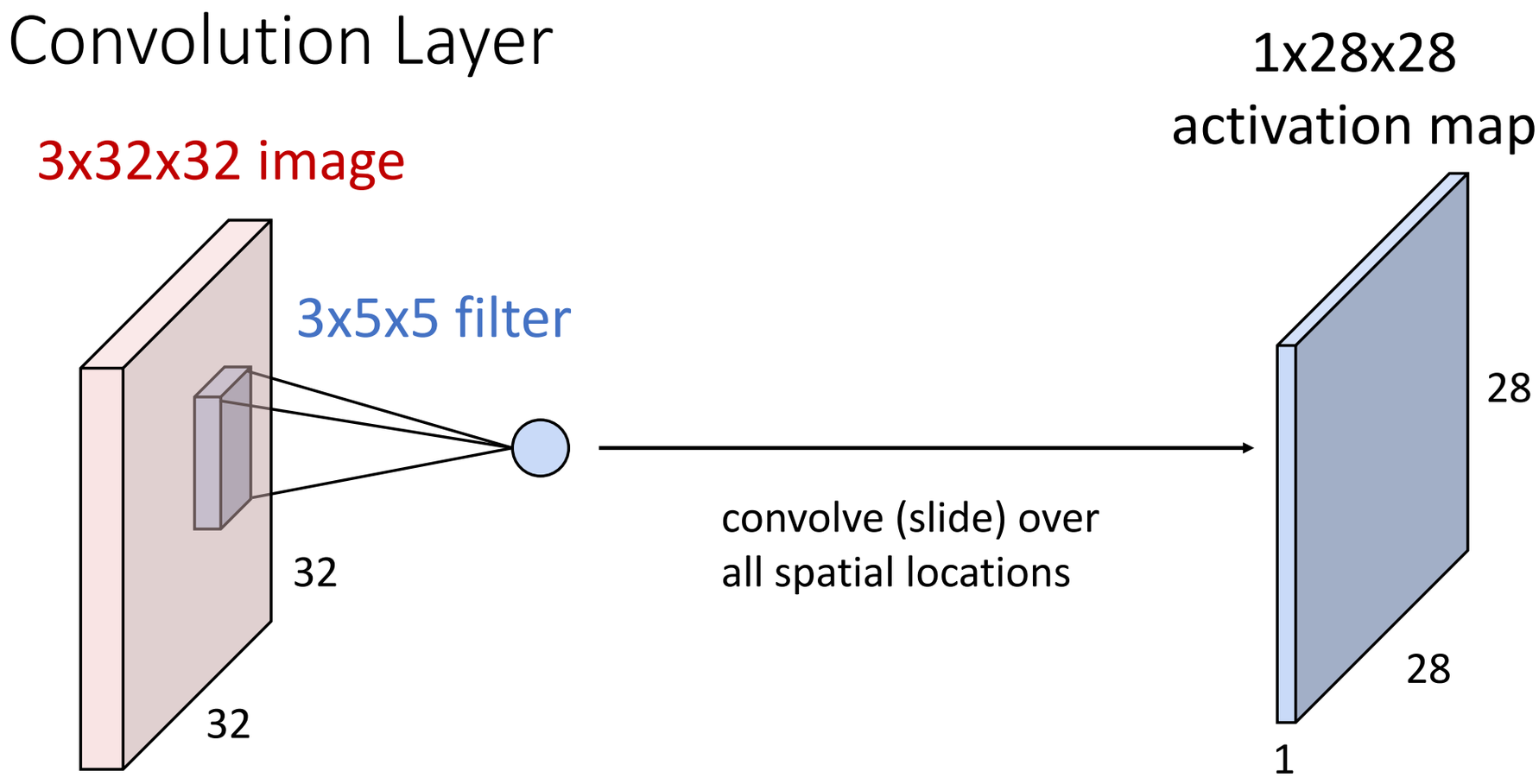


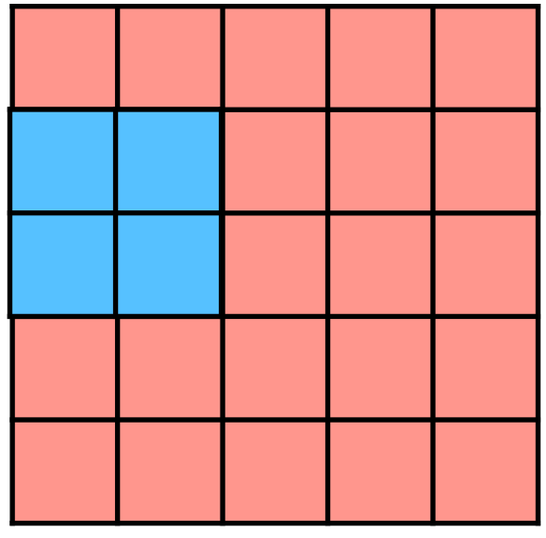
Convolution Layer



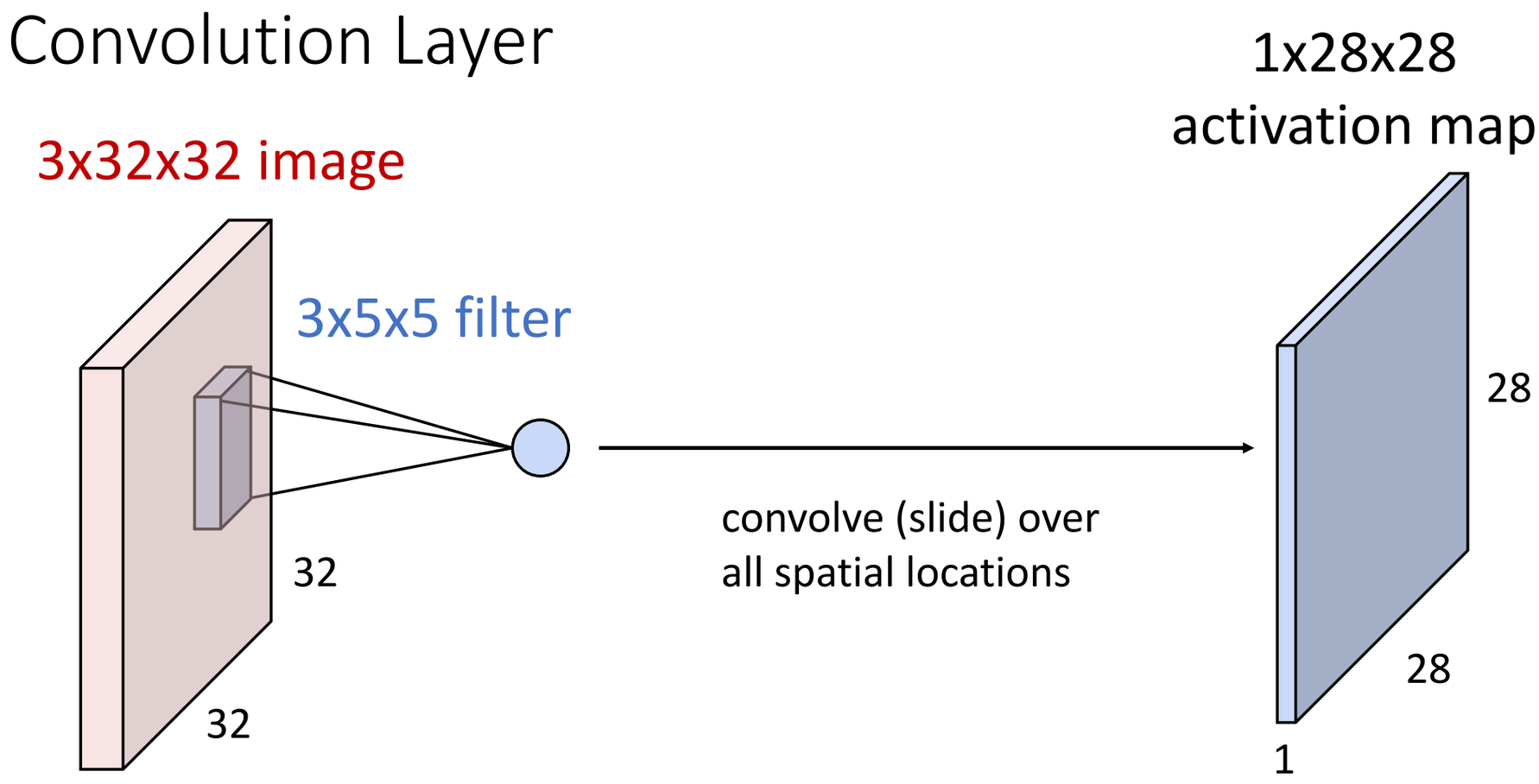


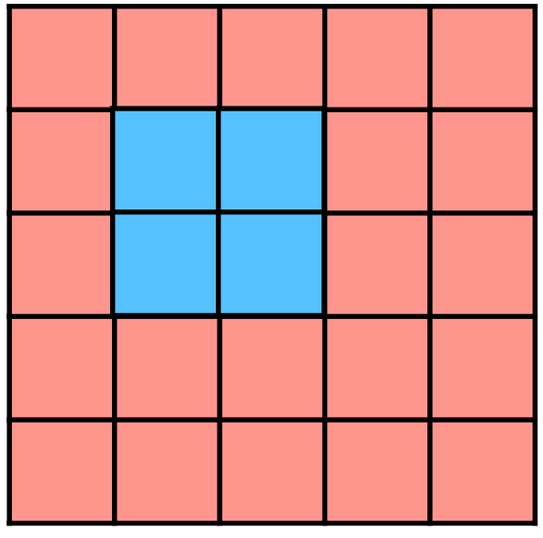
Convolution Layer



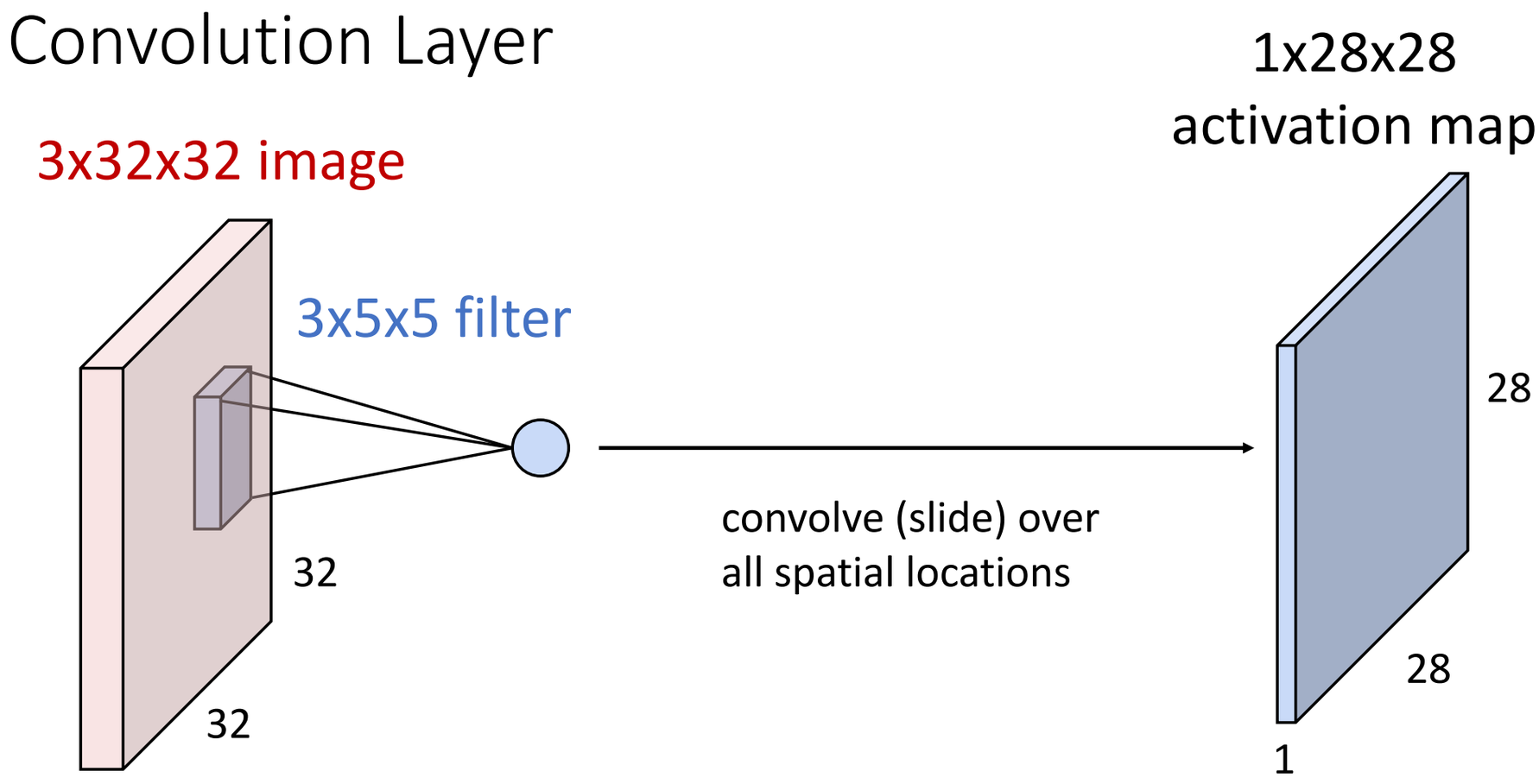


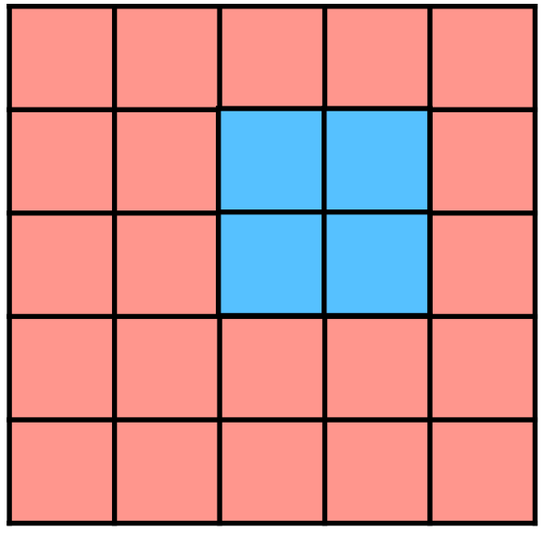
Convolution Layer



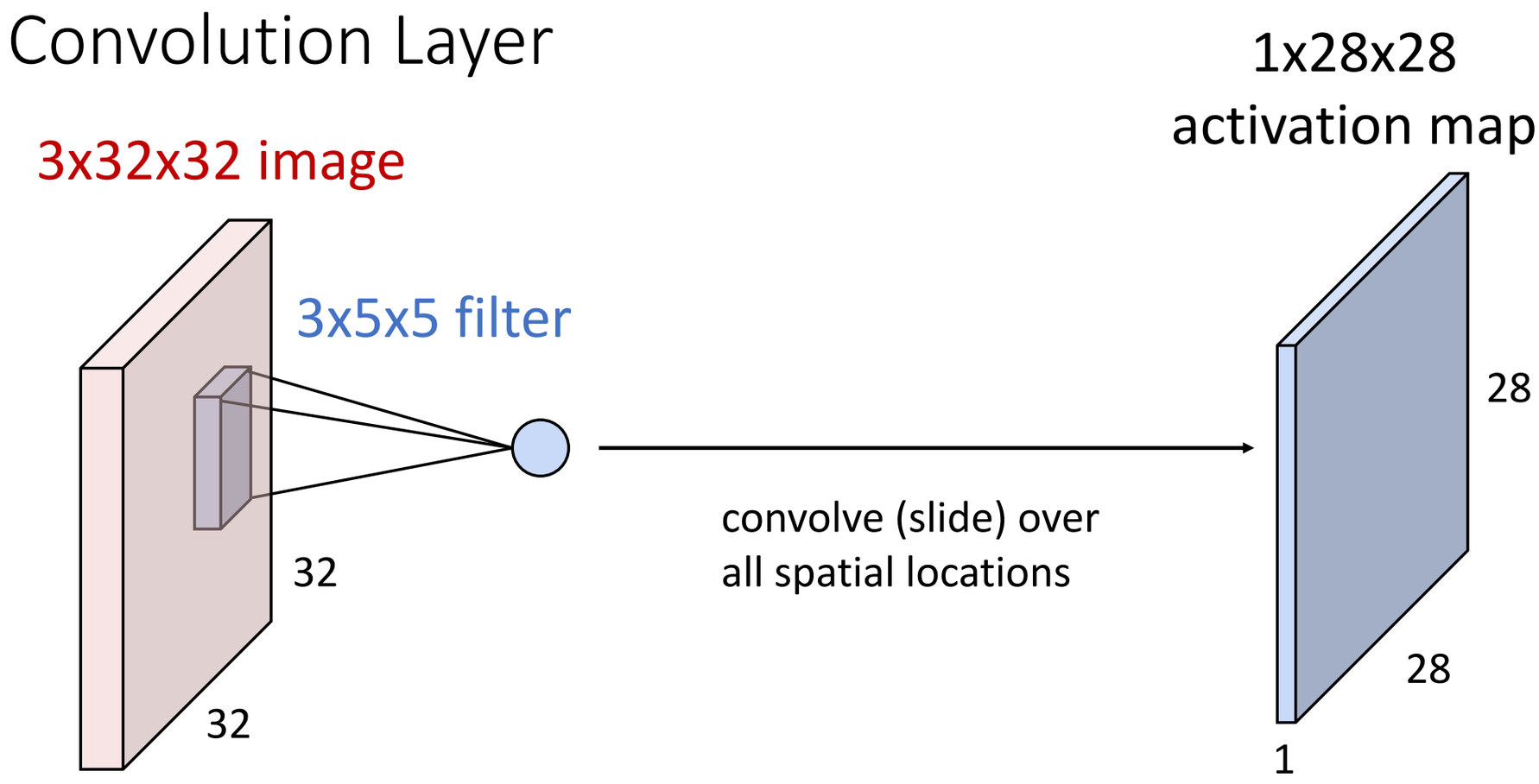


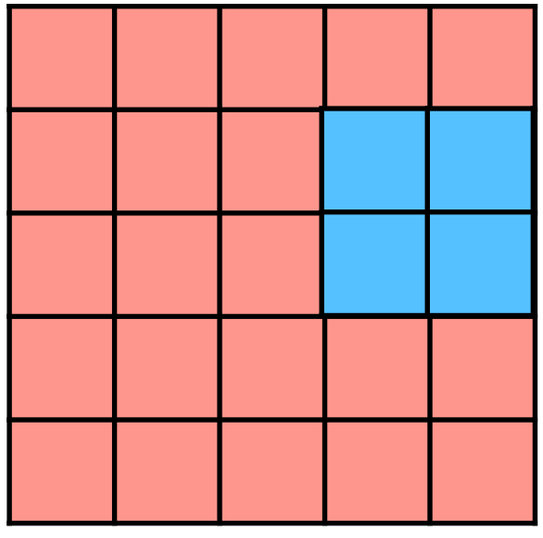
Convolution Layer



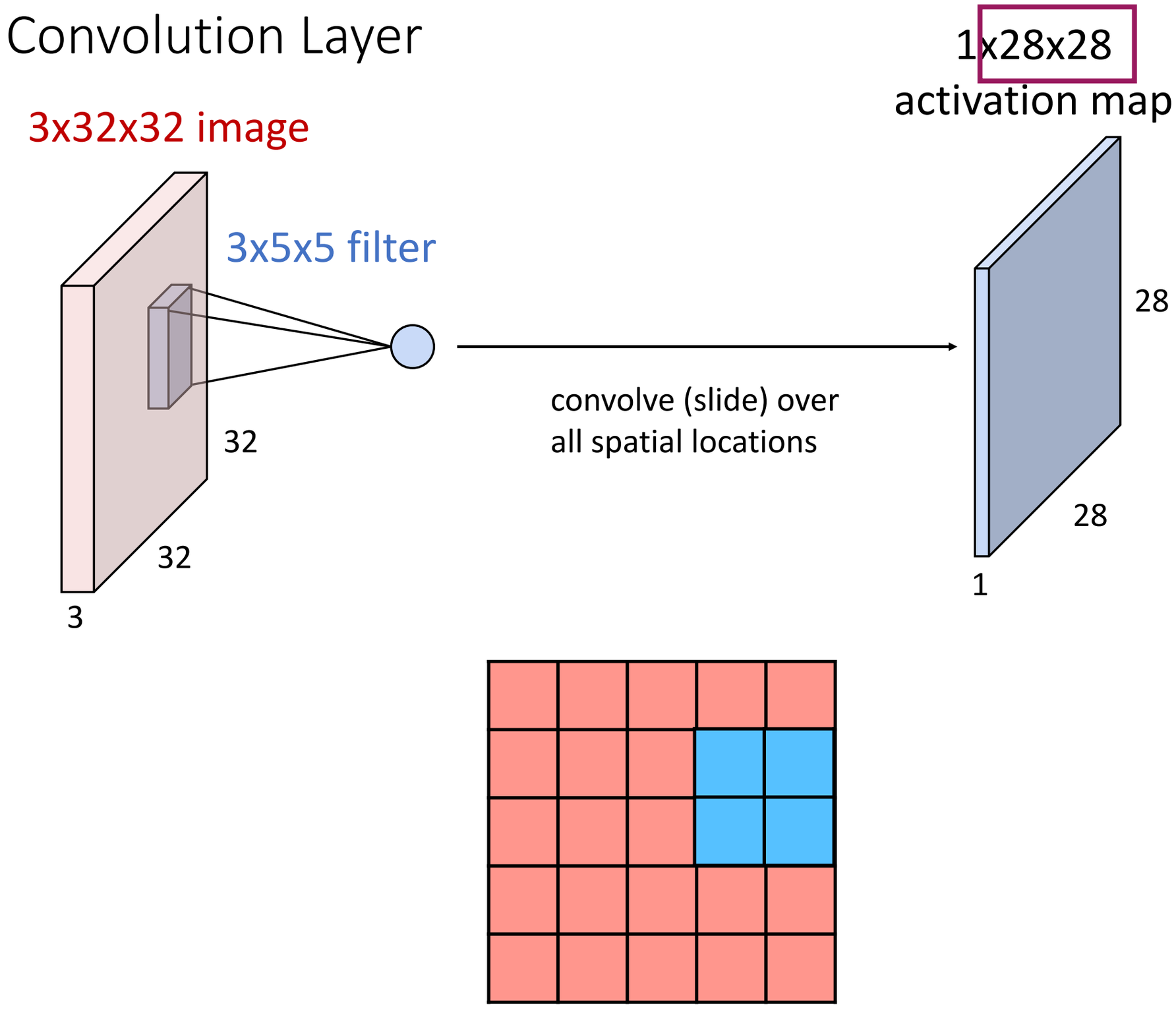


Convolution Layer



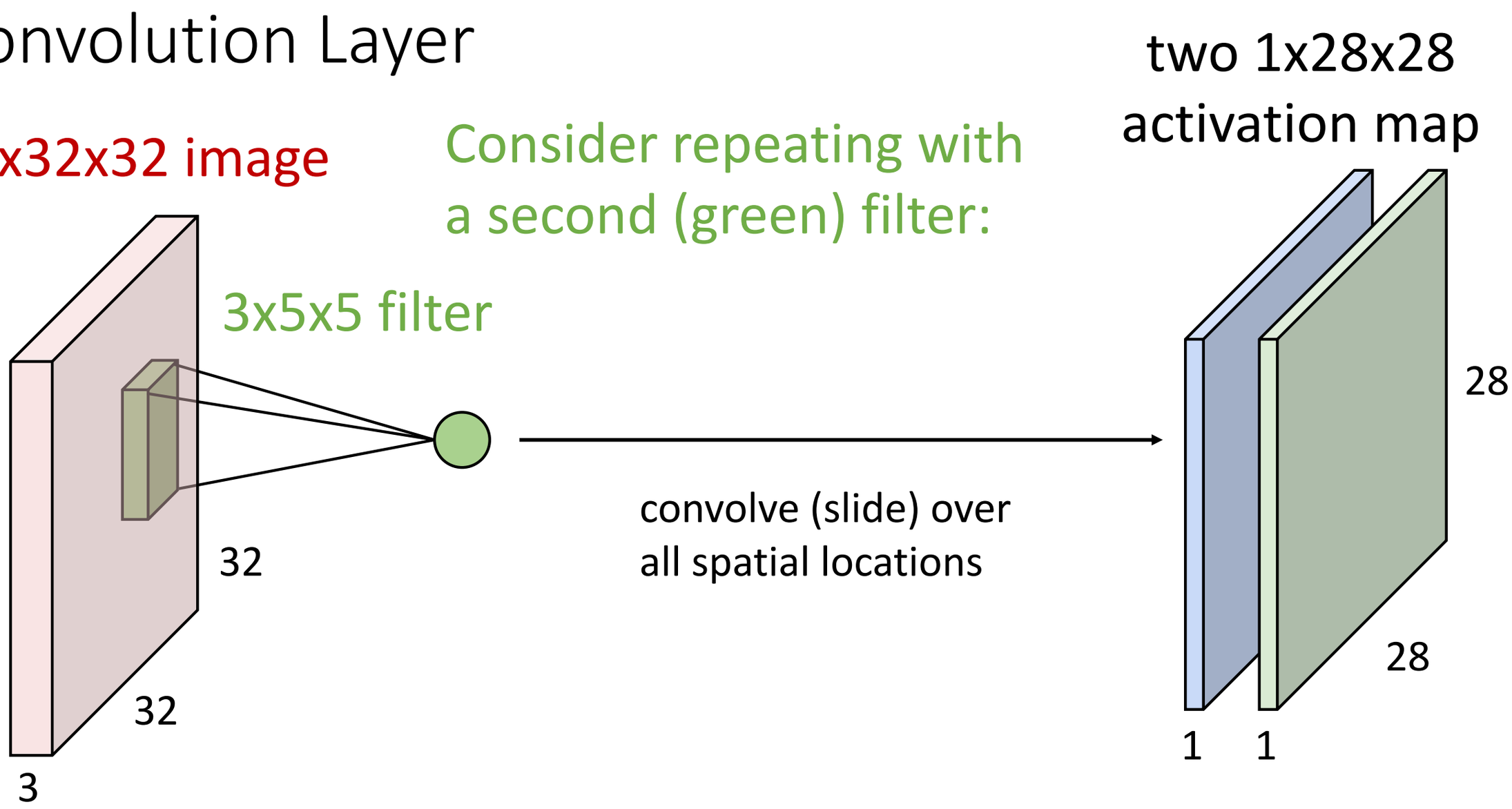


Convolution Layer



Convolution Layer

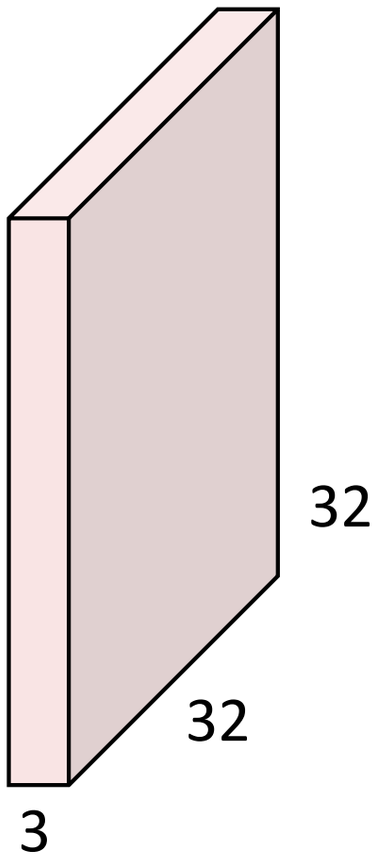
Convolution Layer



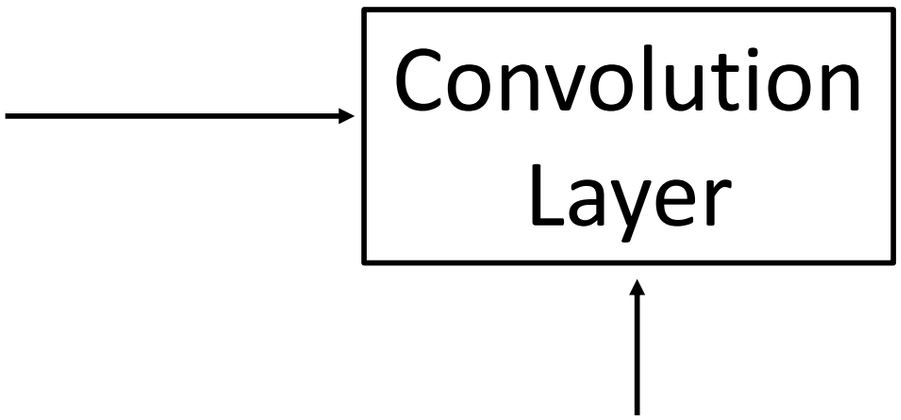
Convolution Layer

Convolution Layer

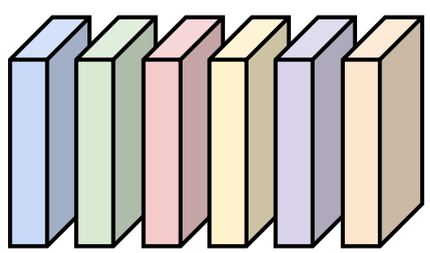
3x32x32 image



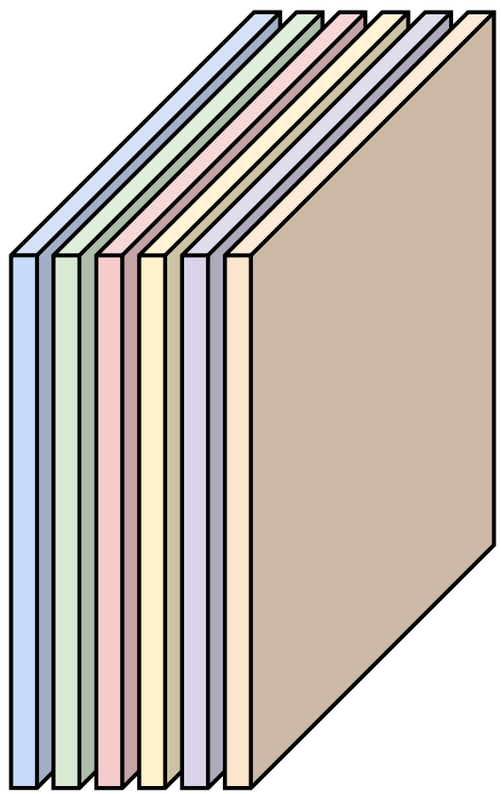
Consider 6 filters,
each 3x5x5



6x3x5x5
filters



6 activation maps,
each 1x28x28

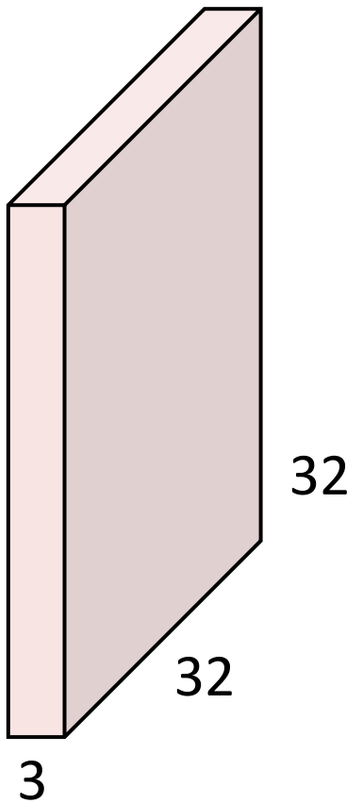


Stack activations to get a
6x28x28 output image!

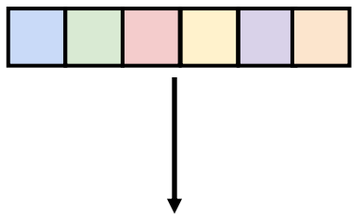
Convolution Layer

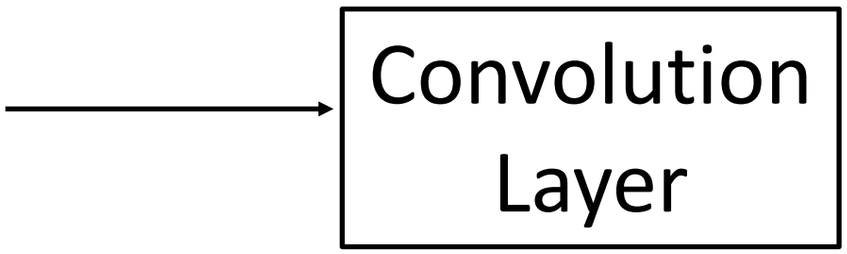
Convolution Layer

3x32x32 image

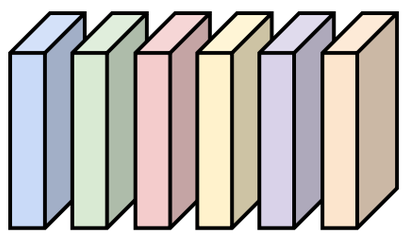


Also 6-dim bias vector:

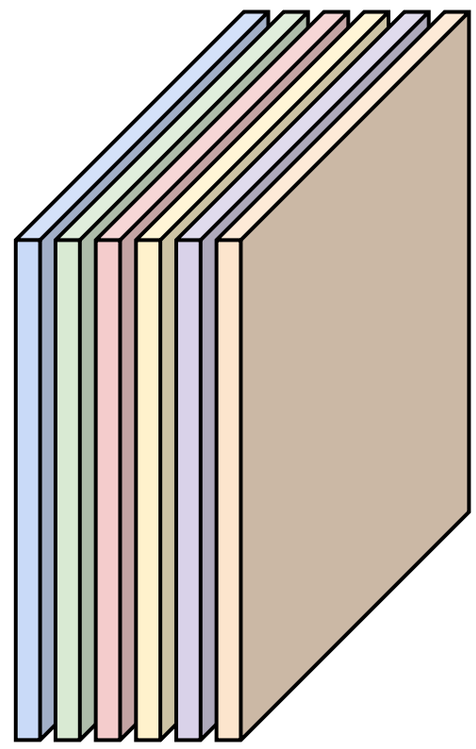




6x3x5x5 filters

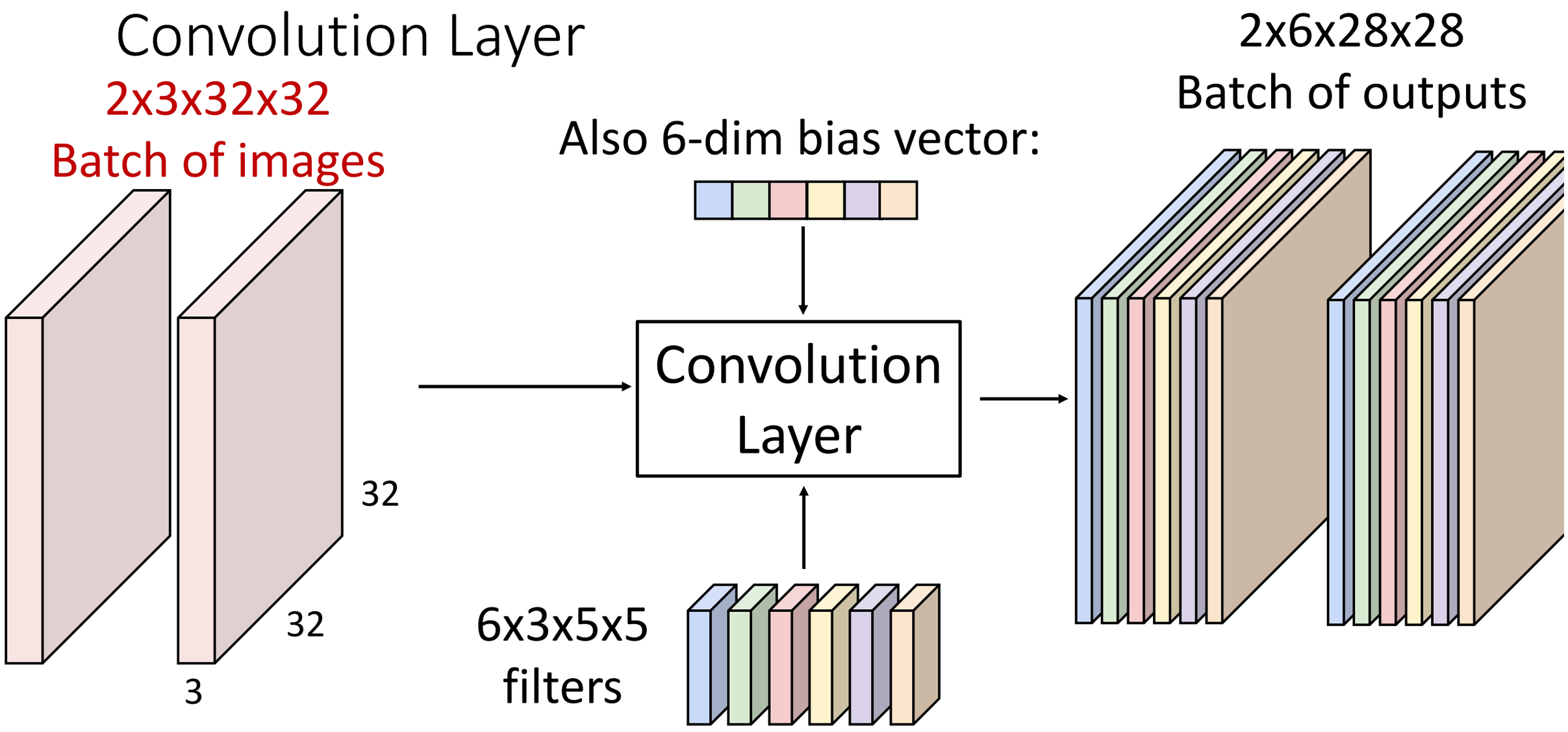


6 activation maps, each 1x28x28

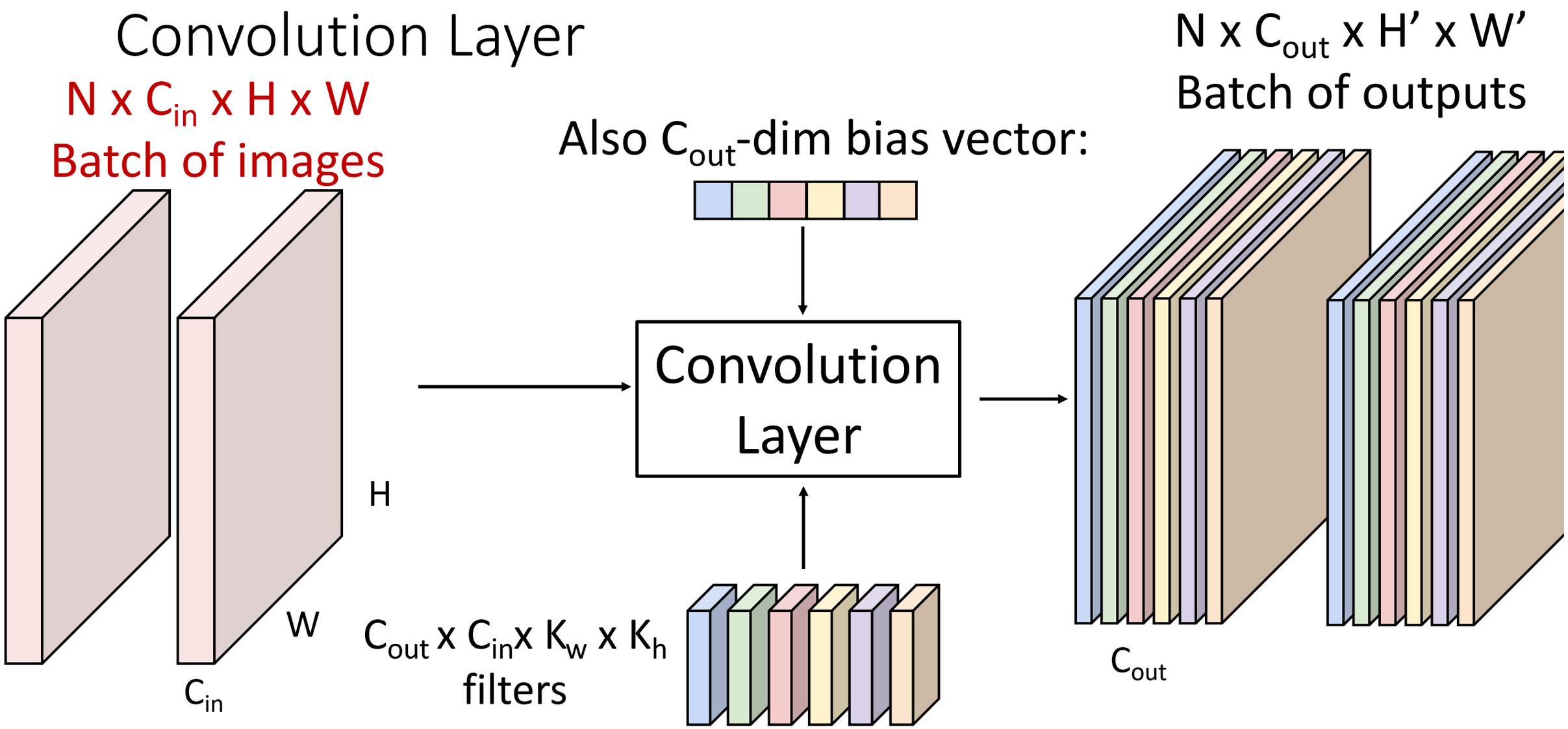


Stack activations to get a 6x28x28 output image!

Convolution Layer

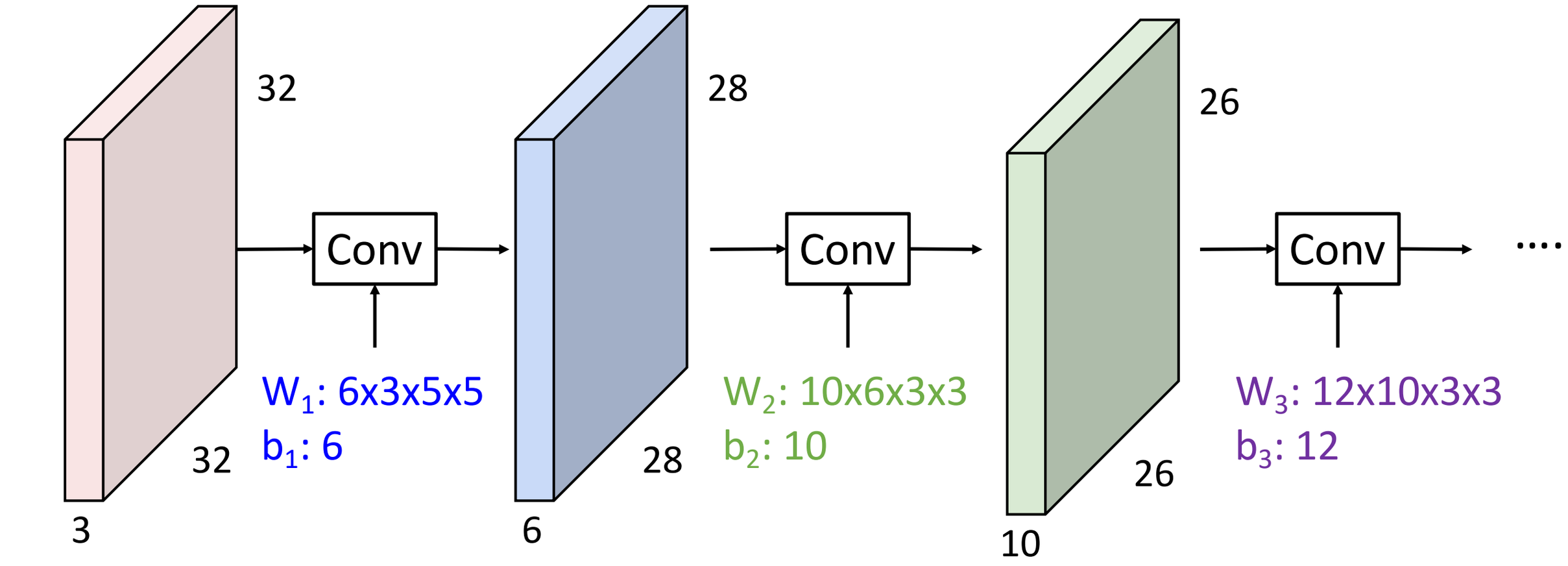


Convolution Layer



Convolution Layer

Stacking Convolutions



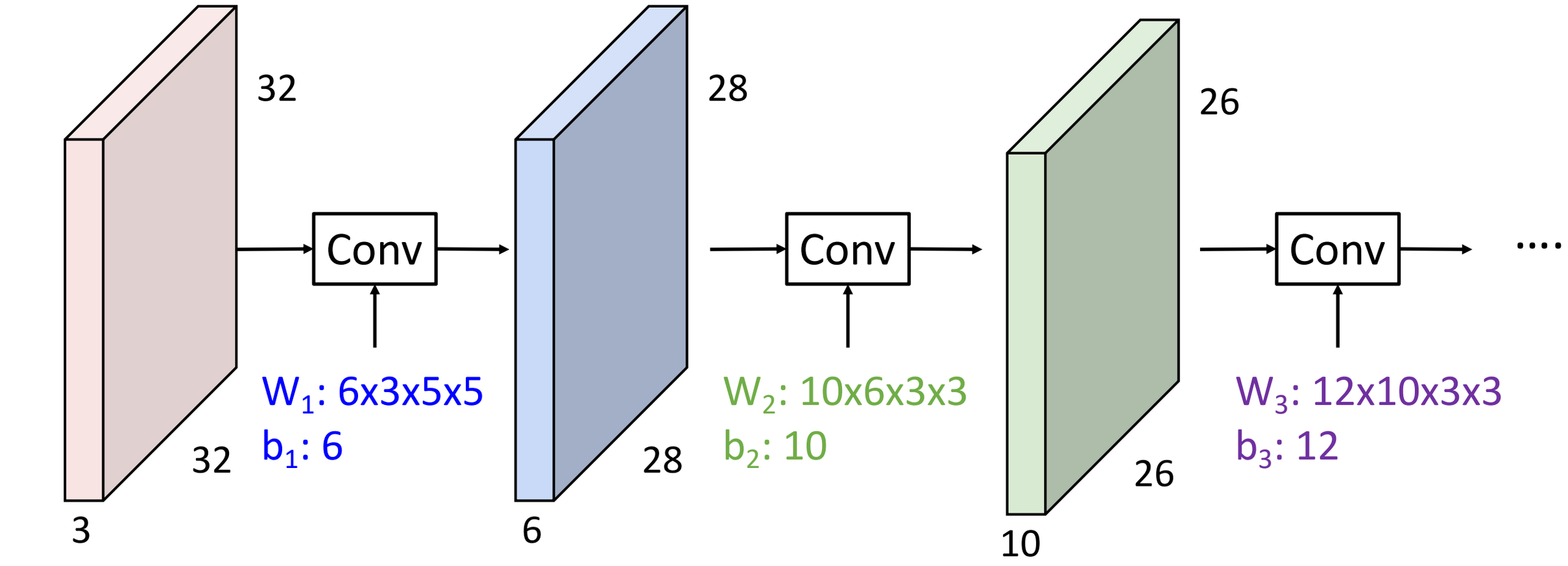
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Convolution Layer

Stacking Convolutions



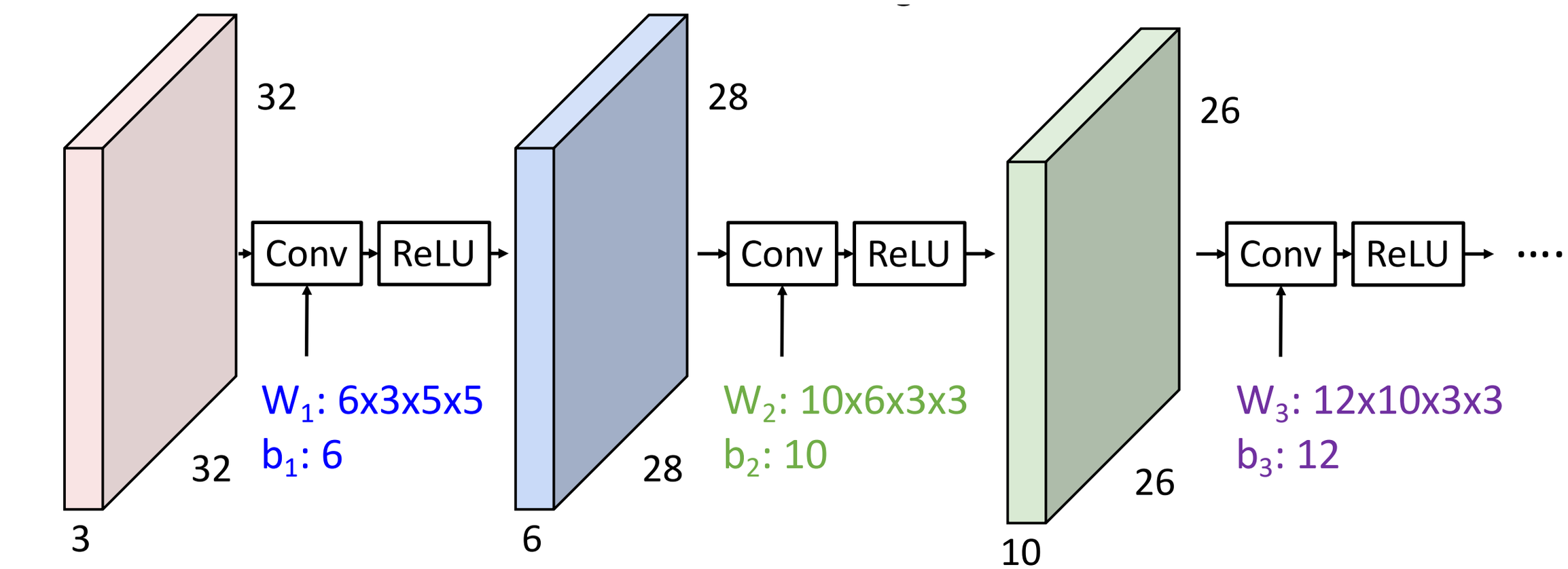
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Convolutions are linear transformations.
Need to add activation functions to obtain non-linearity.

Convolution Layer



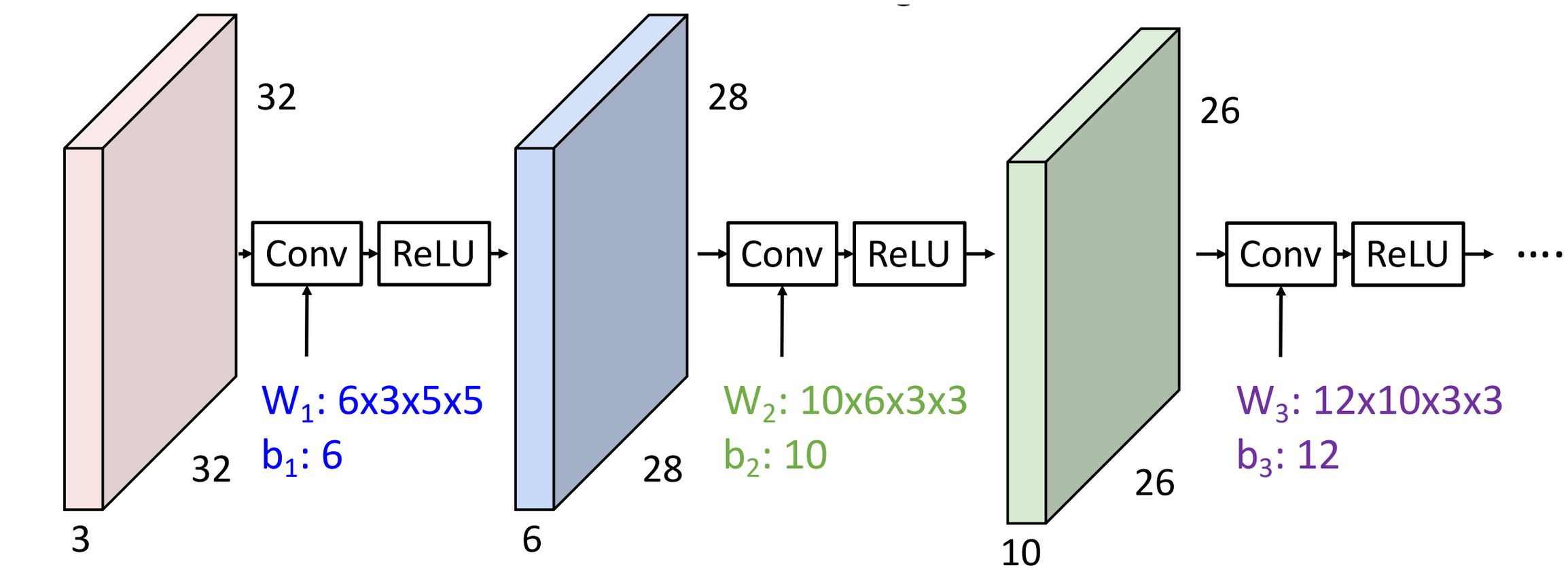
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Convolutions are linear transformations.
Need to add activation functions to obtain non-linearity.

Convolution Layer



Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

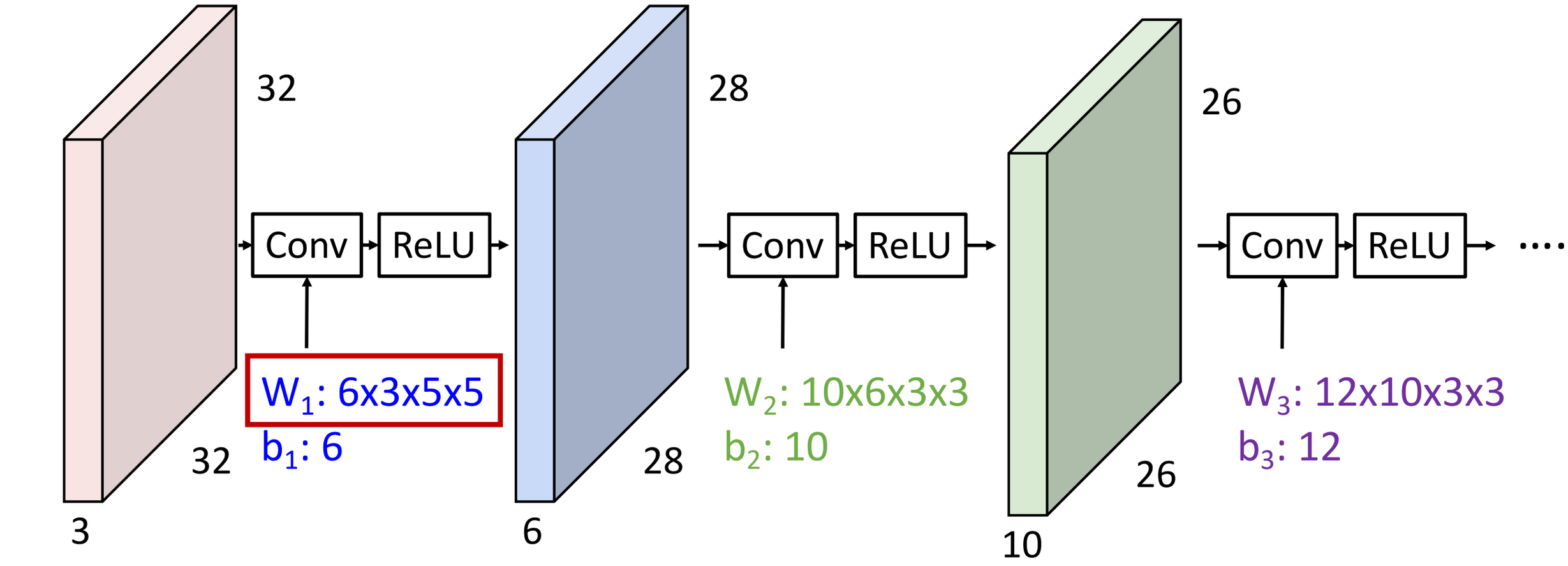
Second hidden layer:
N x 10 x 26 x 26

Convolutions are linear transformations.
Need to add activation functions to obtain non-linearity.

Much fewer parameters than fully-connected layer!

Convolution Layer

What do convolutional filters learn?



Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Linear Transformations are Templates

Linear classifier: One template per class

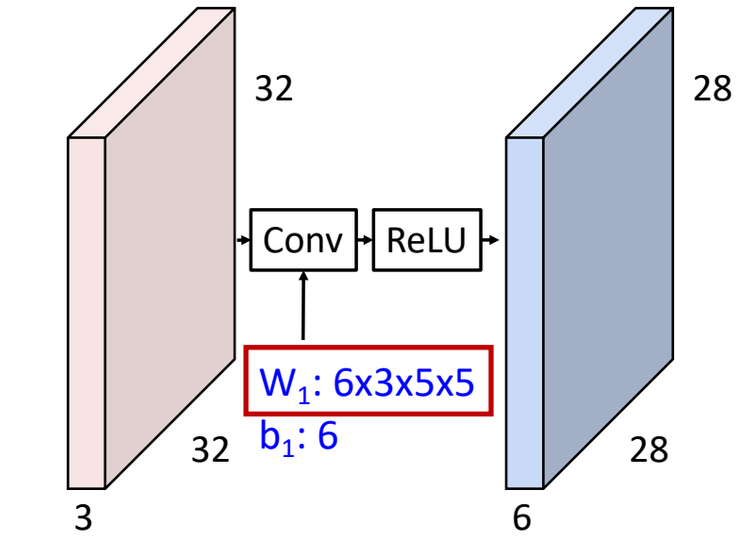
Linear Transformations are Templates

Linear classifier: One template per class

MLP: Bank of whole-image templates

Linear Transformations are Templates

Linear classifier: One template per class



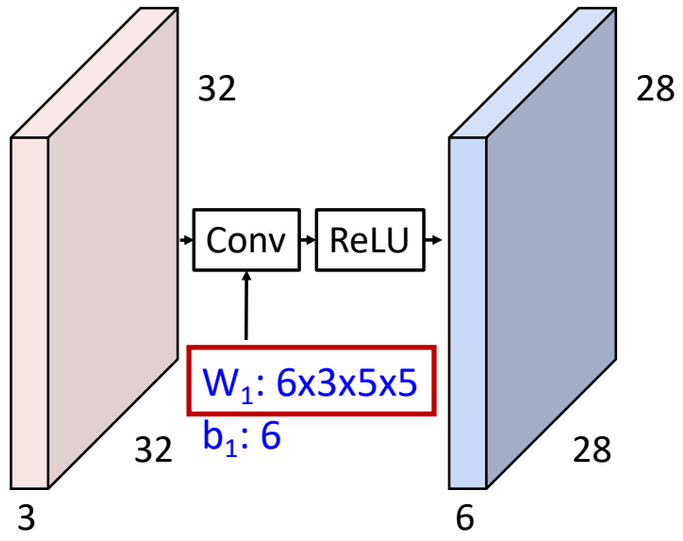
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

MLP: Bank of whole-image templates

Linear Transformations are Templates

Linear classifier: One template per class

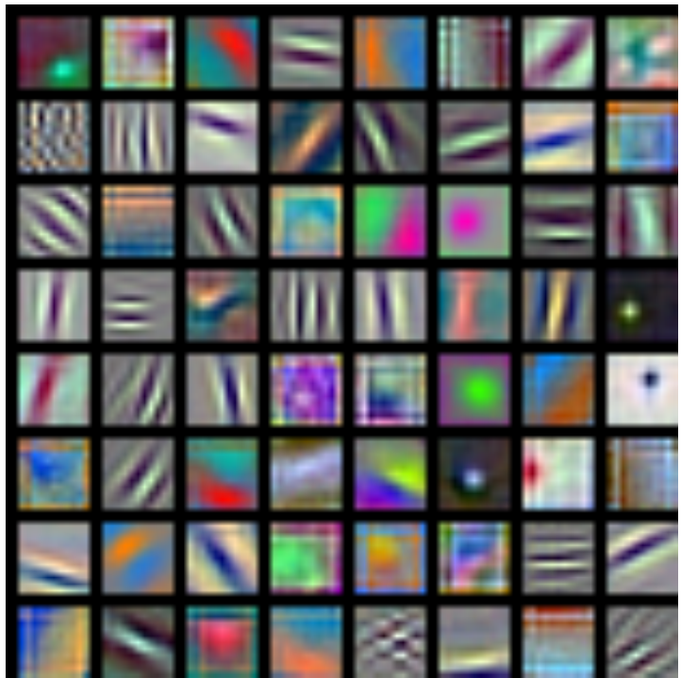


Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

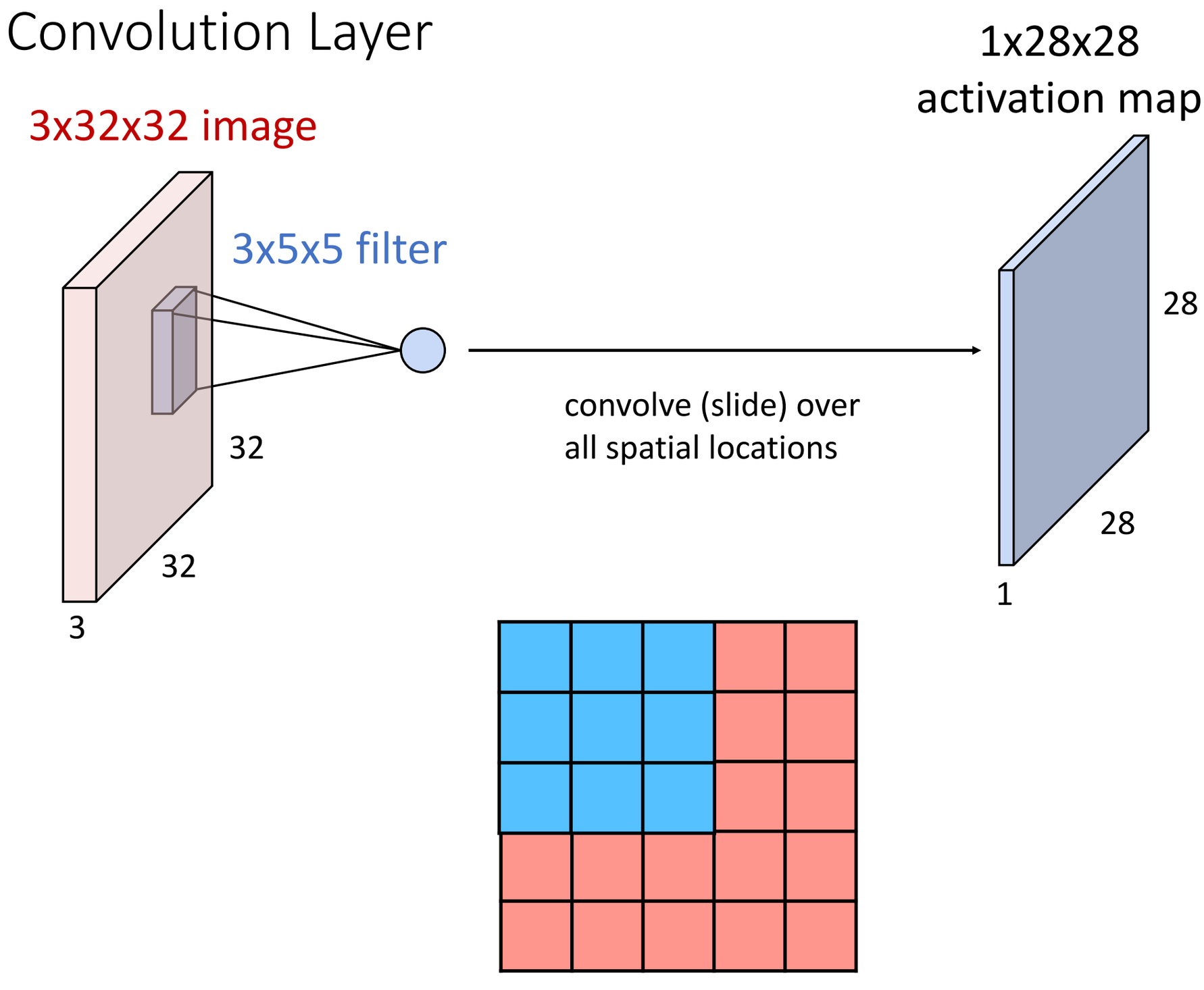
MLP: Bank of whole-image templates

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

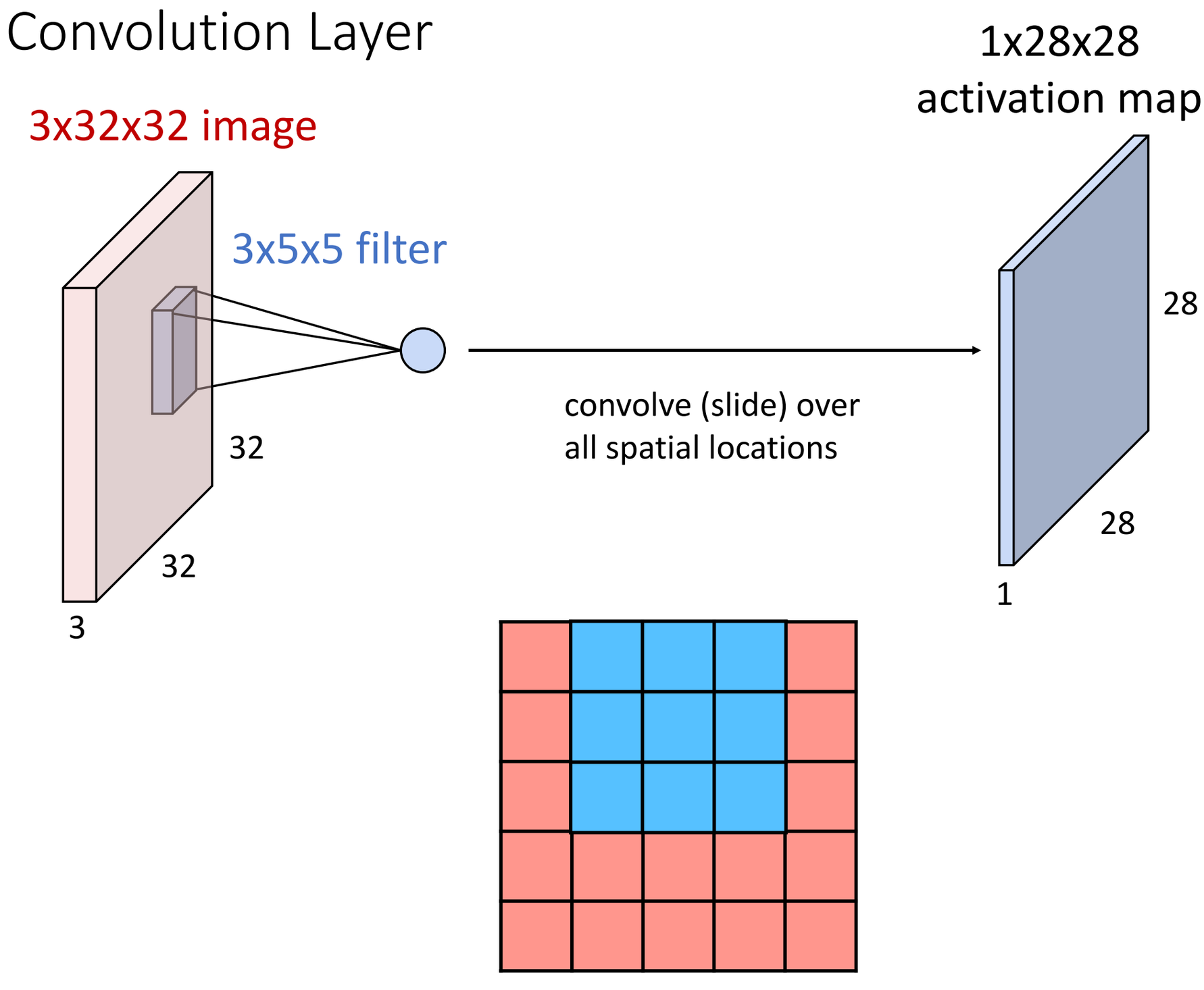


AlexNet: 64 filters, each 3x11x11

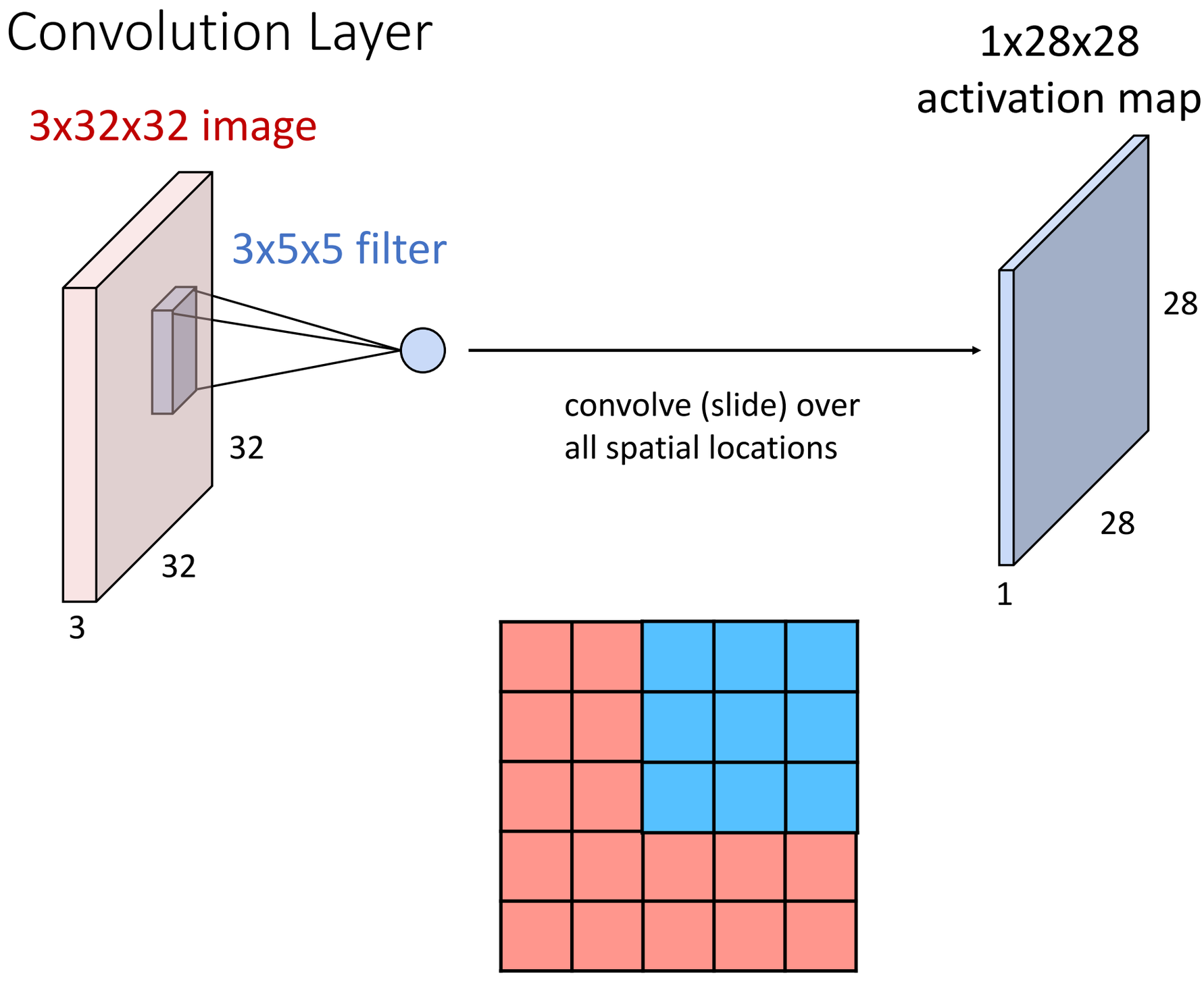
Padding


Convolution Layer
3x32x32 image
3x5x5 filter
32
32
3
convolve (slide) over
all spatial locations
1x28x28
activation map
28
28
1

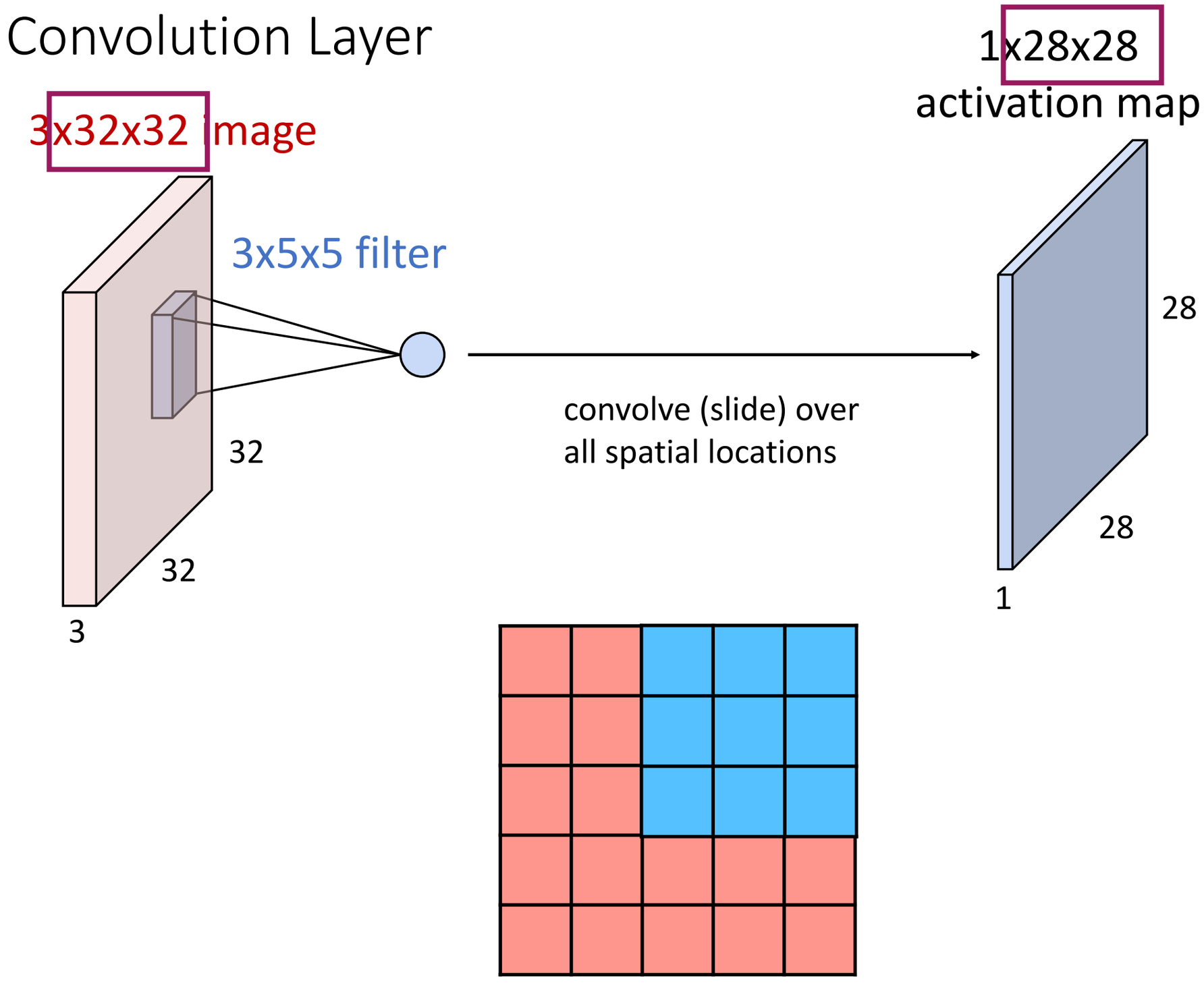
Padding


Convolution Layer
3x32x32 image
3x5x5 filter
32
32
3
convolve (slide) over
all spatial locations
1x28x28
activation map
28
28
1

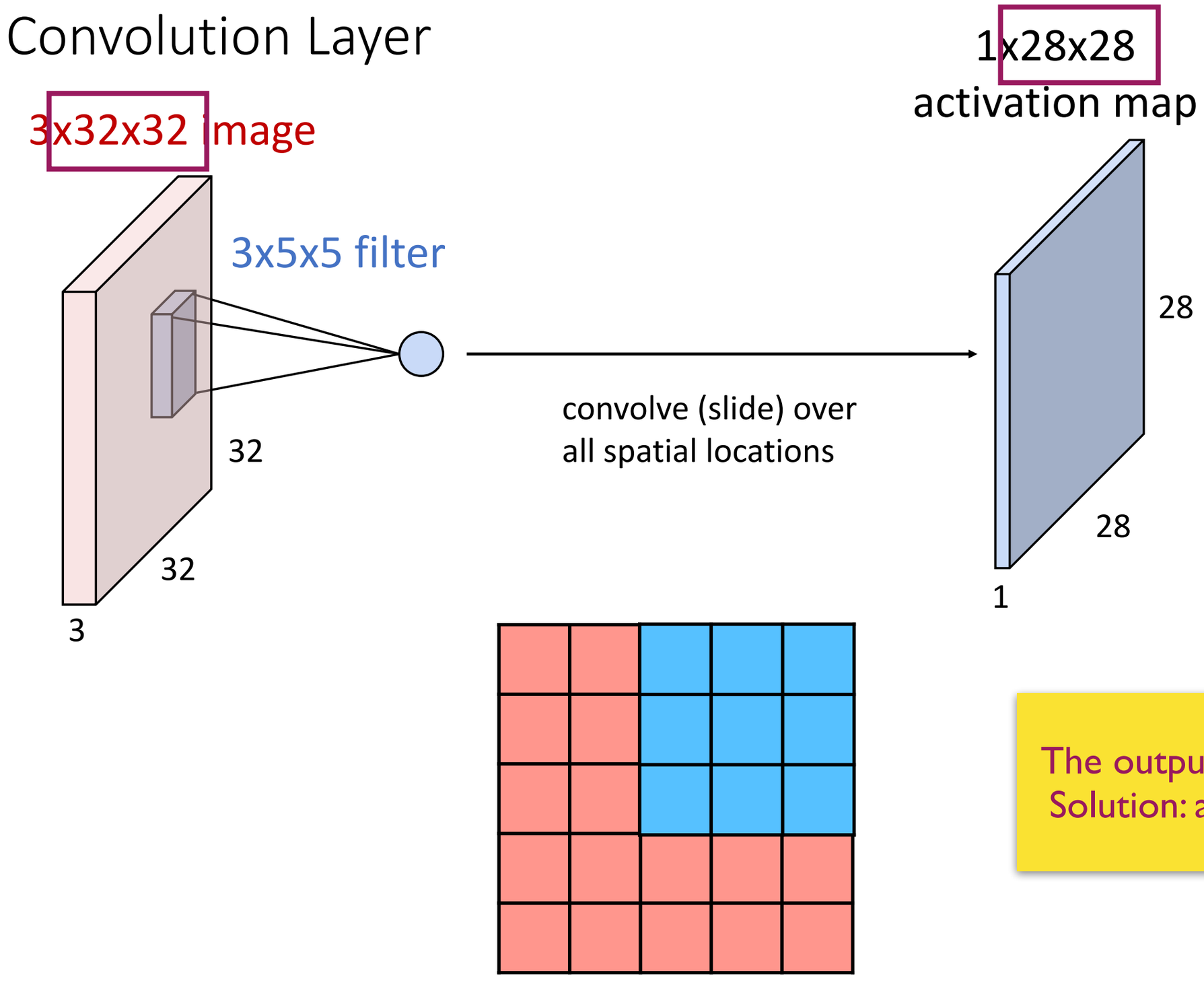
Padding


Convolution Layer
3x32x32 image
3x5x5 filter
32
32
3
convolve (slide) over
all spatial locations
1x28x28
activation map
28
28
1

Padding

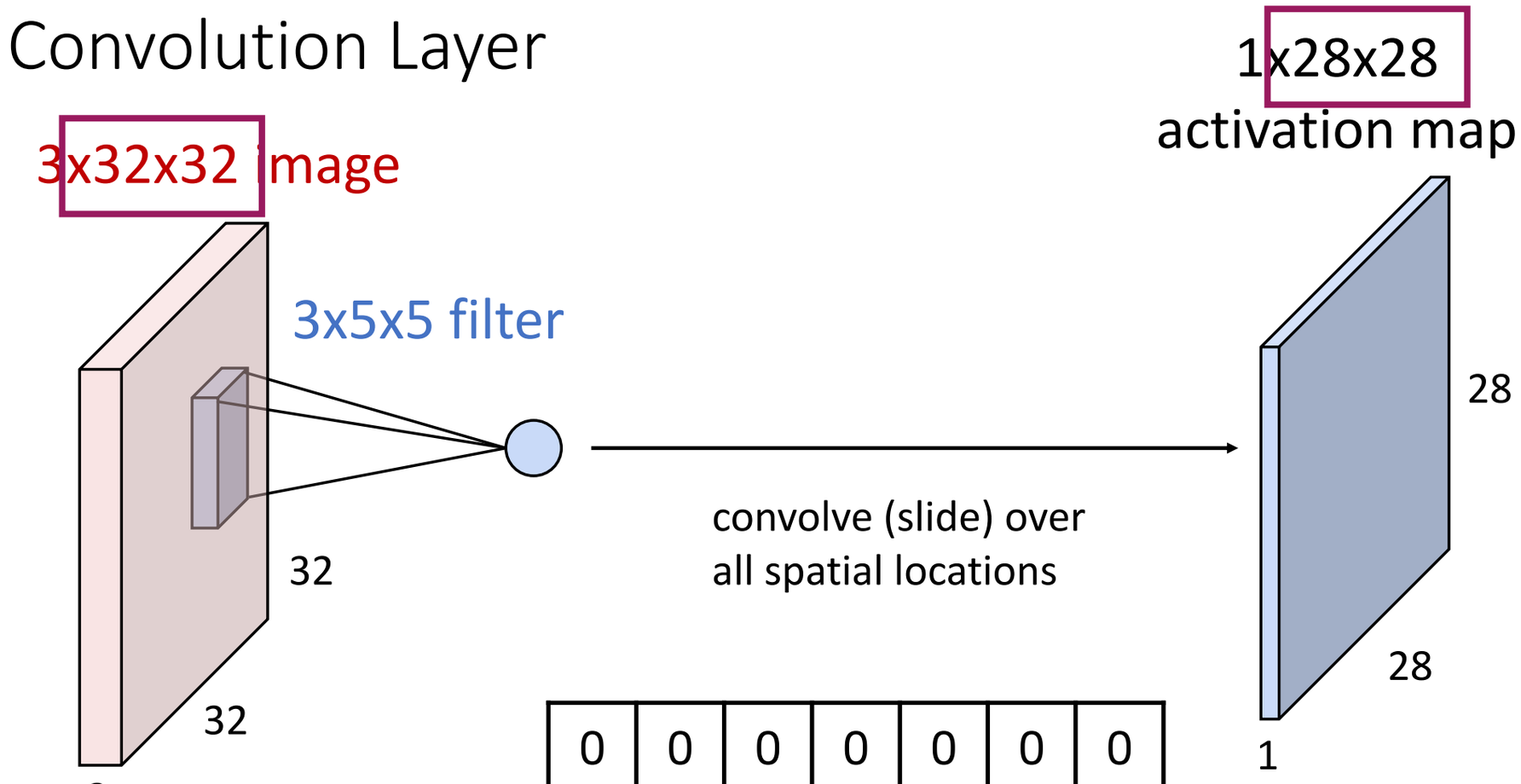

Convolution Layer
3x32x32 image
3x5x5 filter
32
32
3
convolve (slide) over
all spatial locations
1x28x28
activation map
28
28
1

Padding



The output size decreases!
Solution: adding on edges.

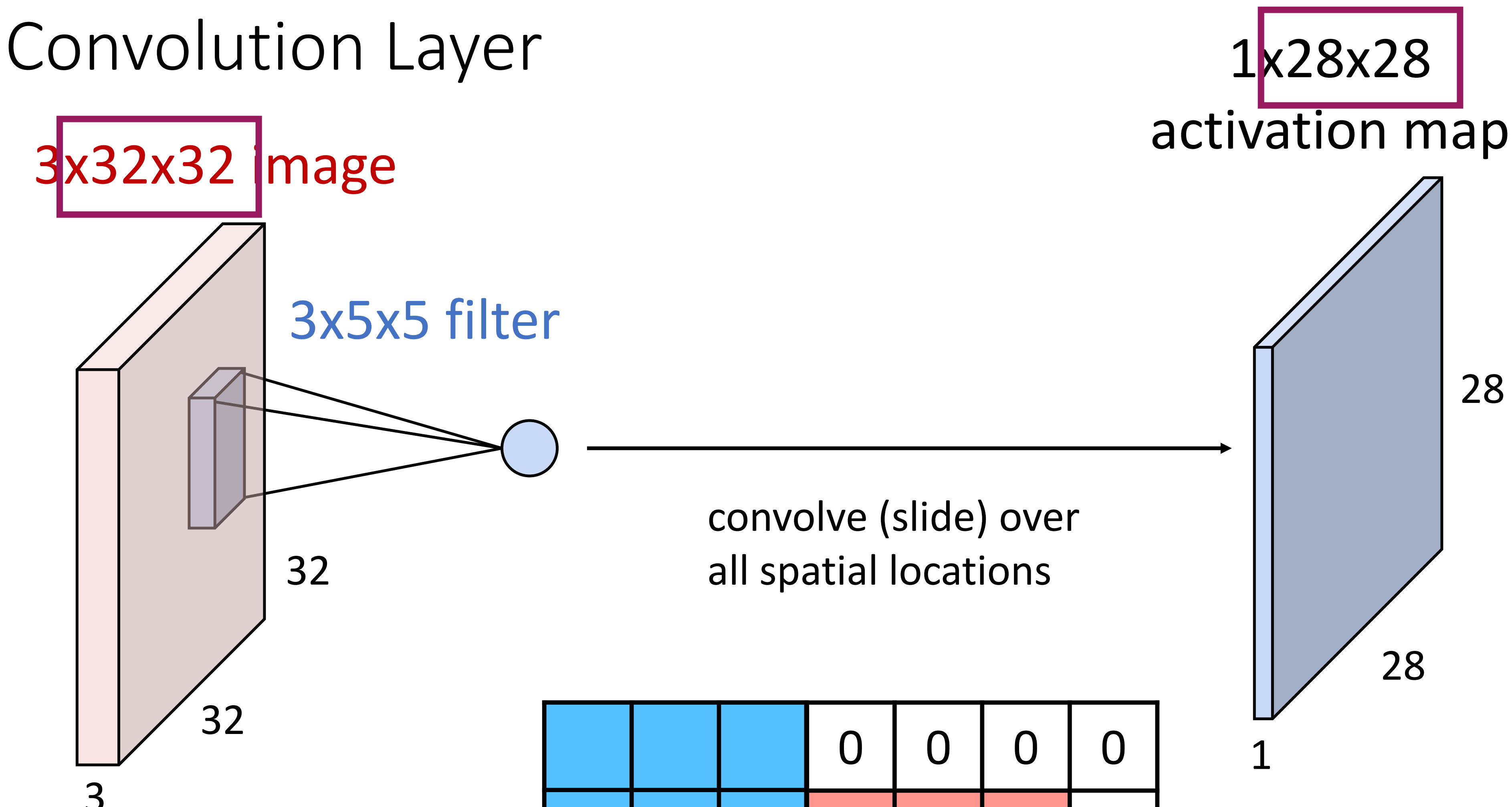
Padding

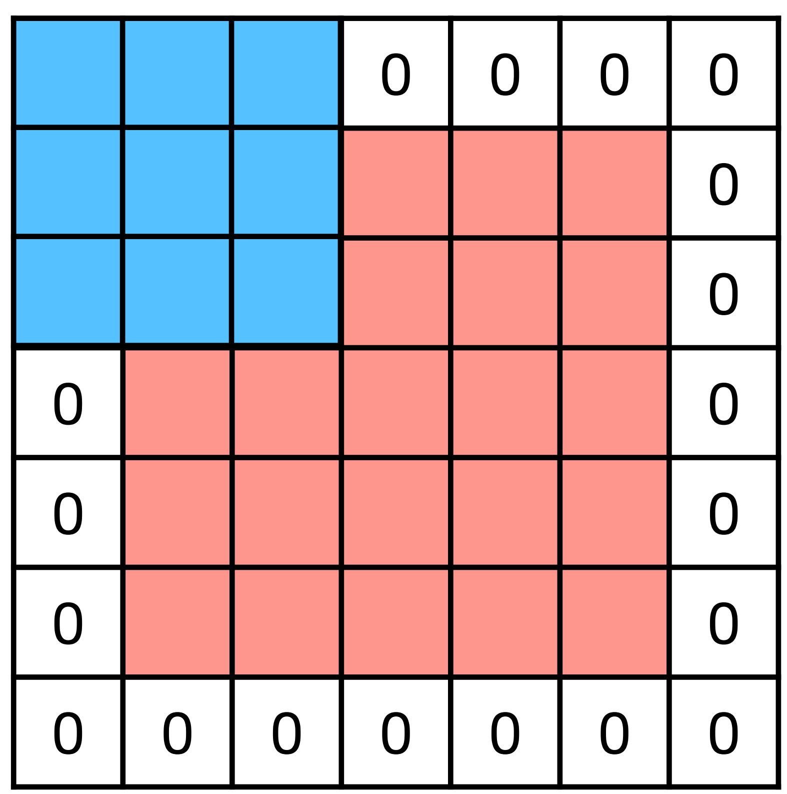


0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

The output size decreases!
Solution: adding on edges.

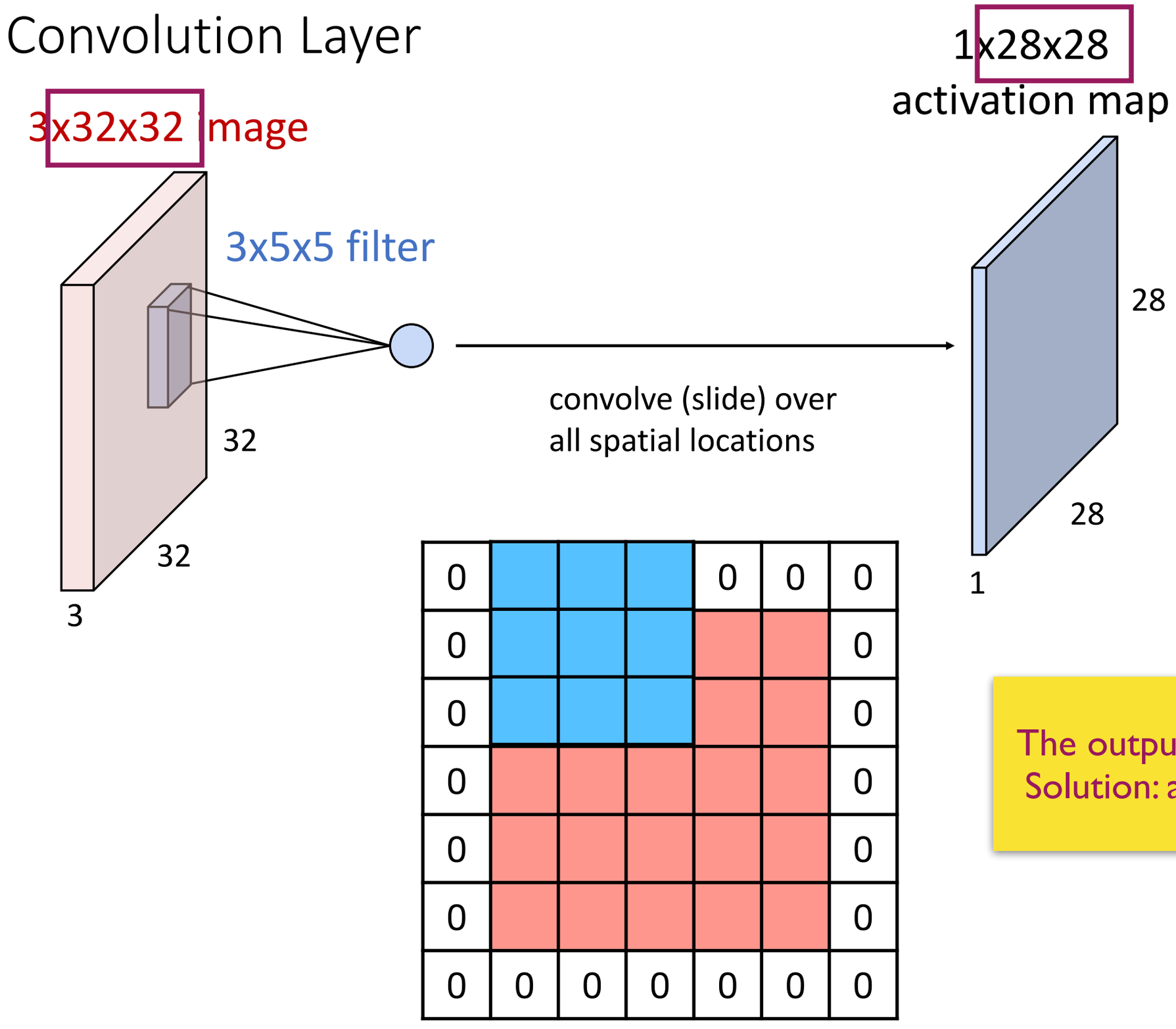
Padding





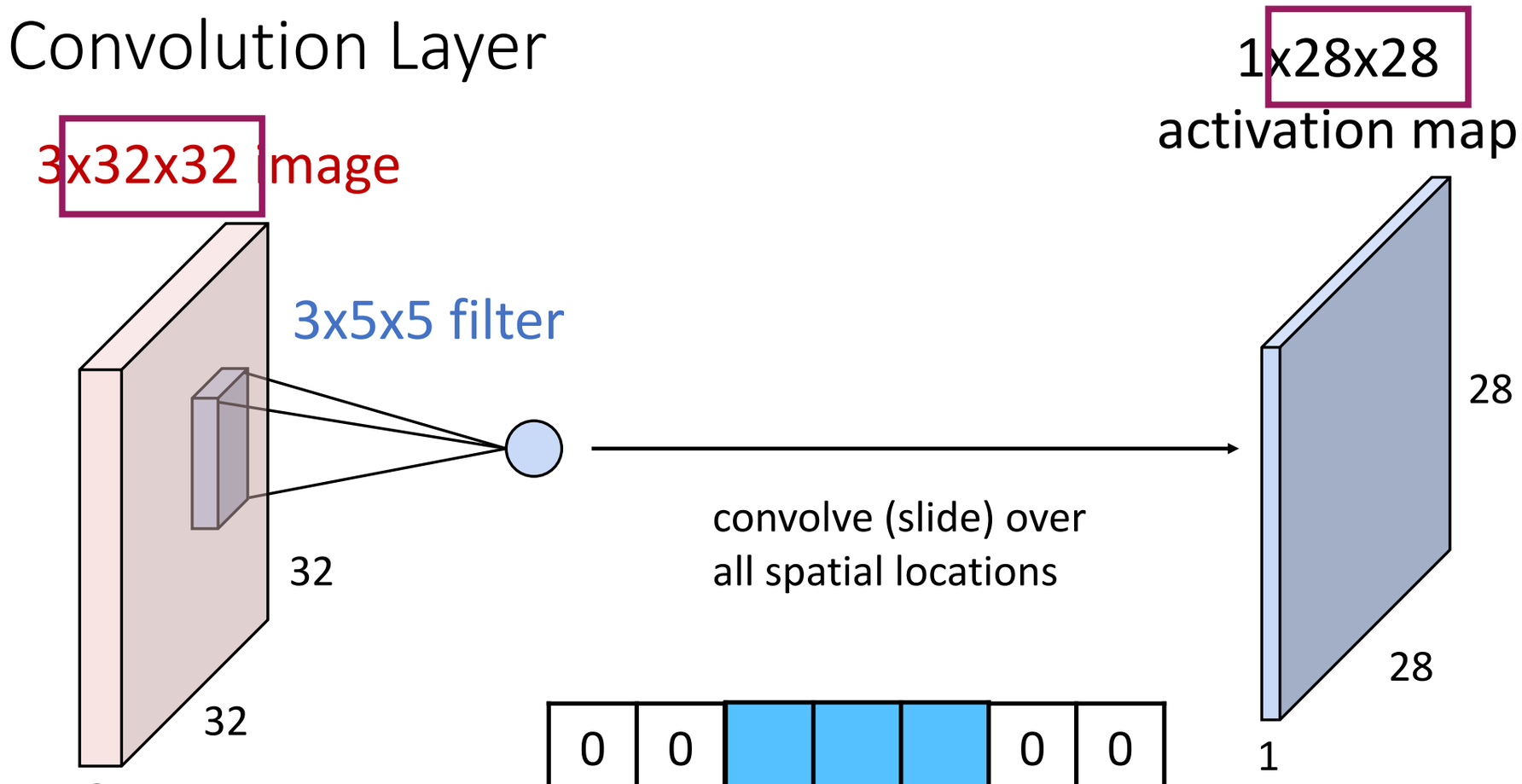
The output size decreases!
Solution: adding on edges.

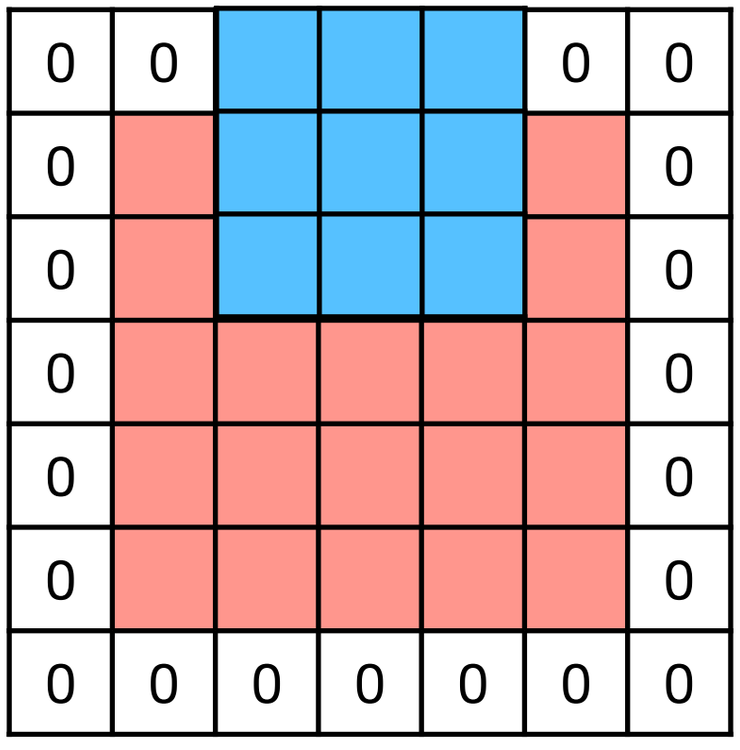
Padding



The output size decreases!
Solution: adding on edges.

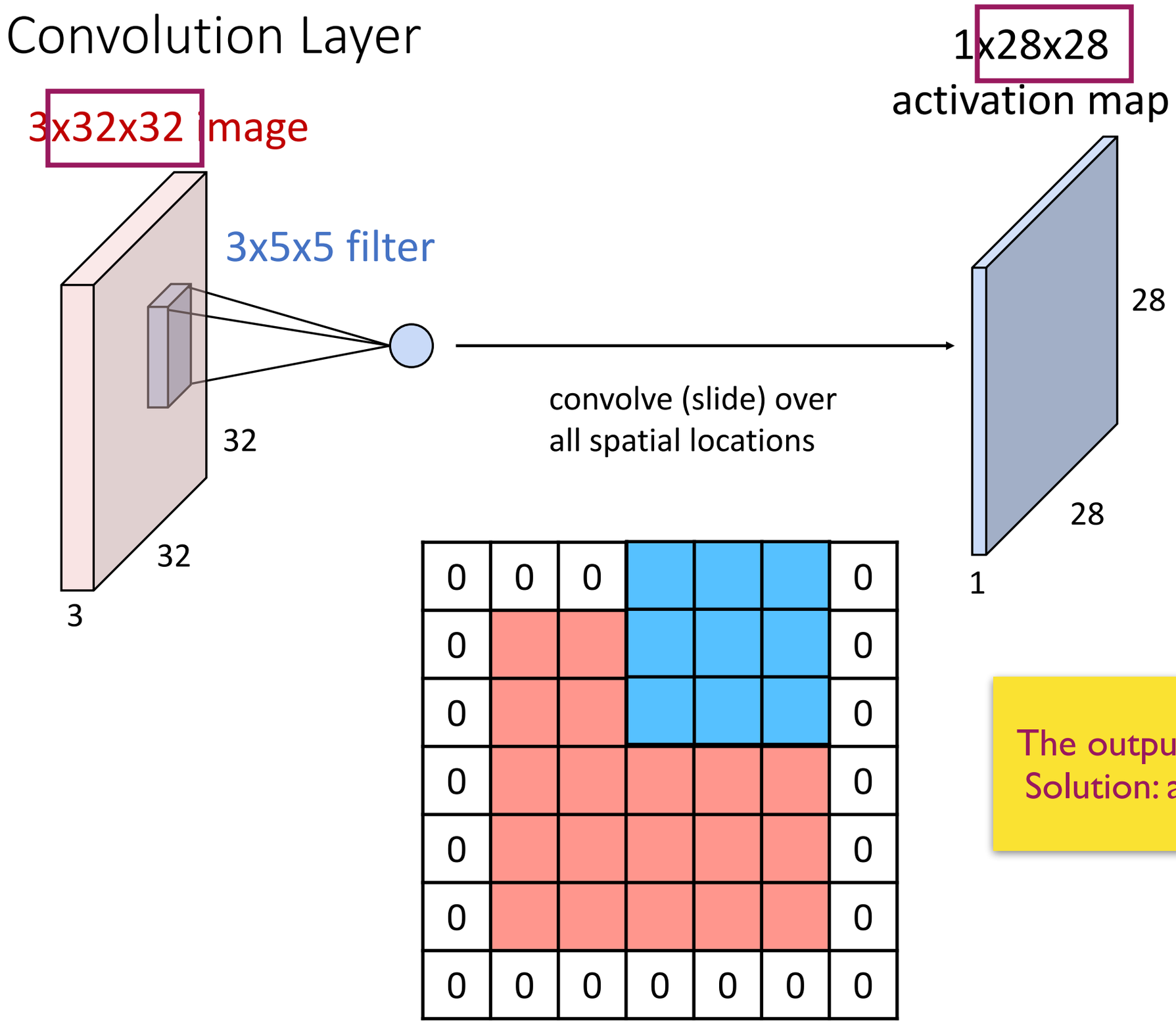
Padding





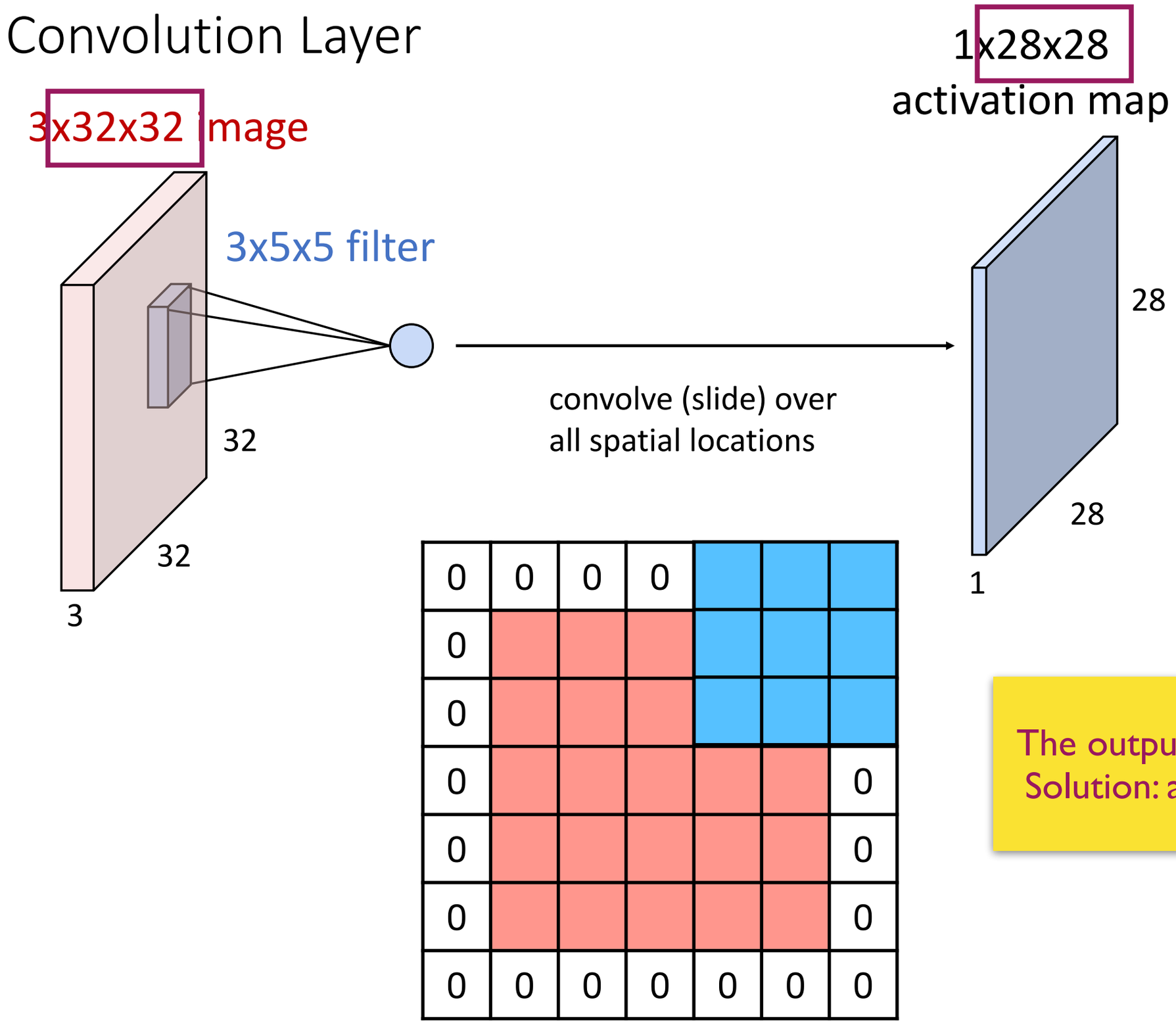
The output size decreases!
Solution: adding on edges.

Padding



The output size decreases!
Solution: adding on edges.

Padding

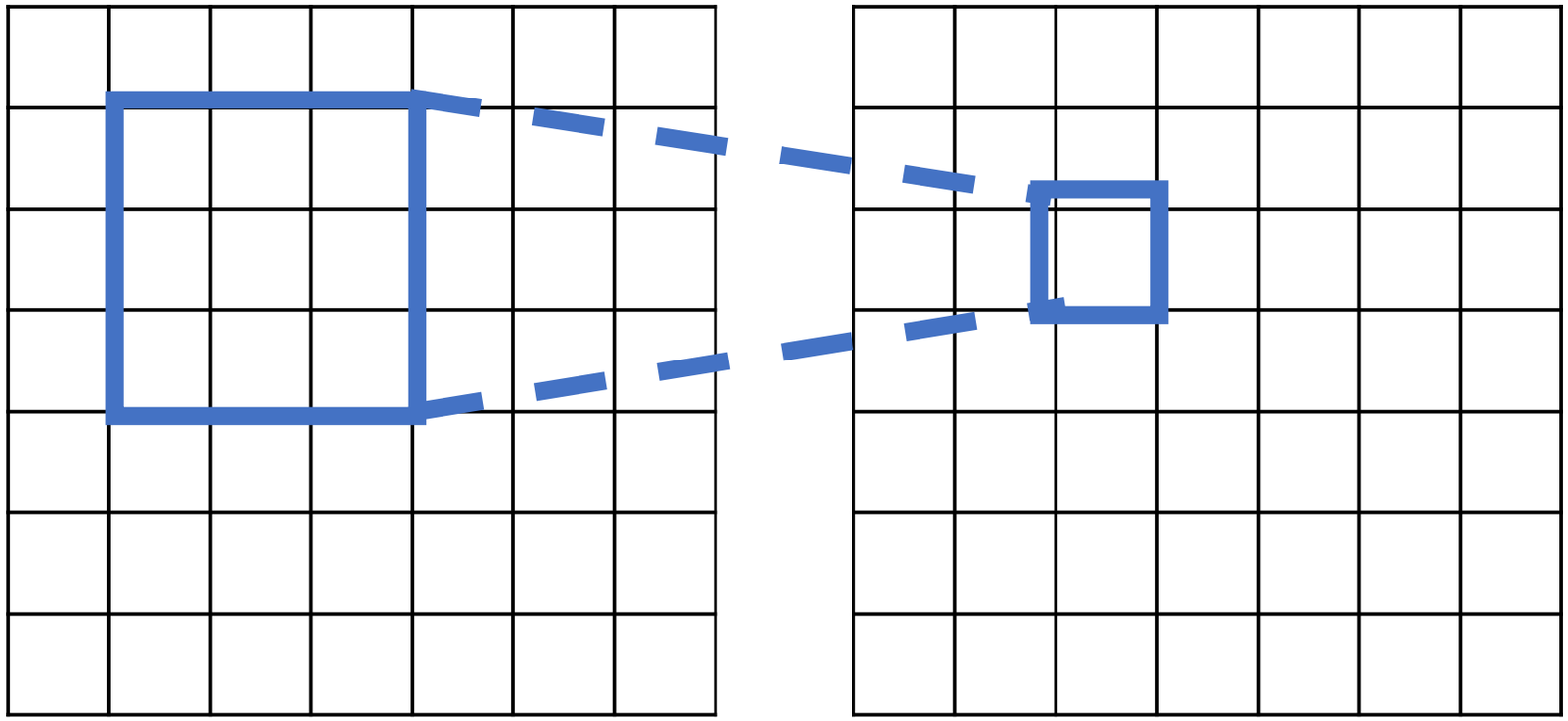


0	0	0	0			
0						
0						
0						0
0						0
0						0
0	0	0	0	0	0	0

The output size decreases!
Solution: adding on edges.

Receptive Fields

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input

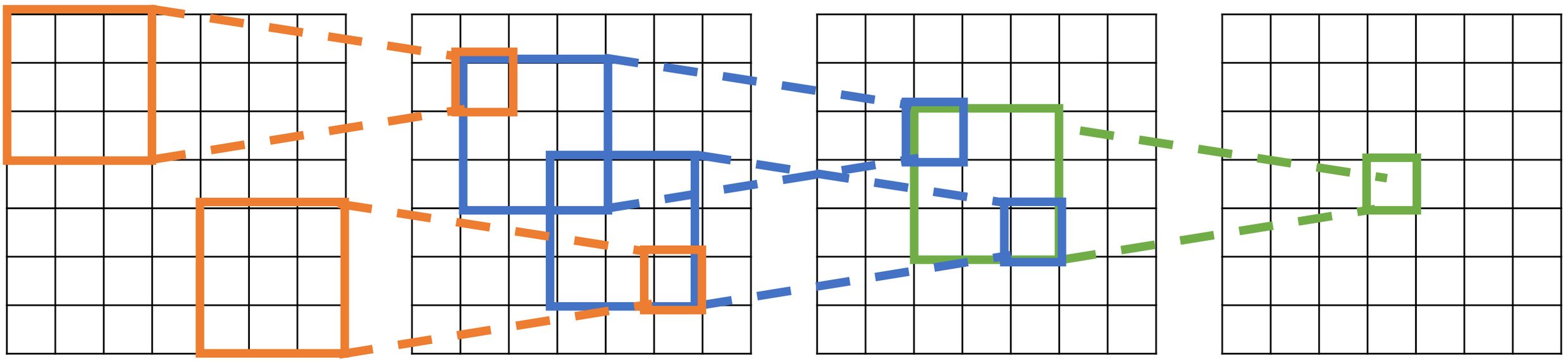


Input Output

Each unit in the output only corresponds a local area of the input.

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Input

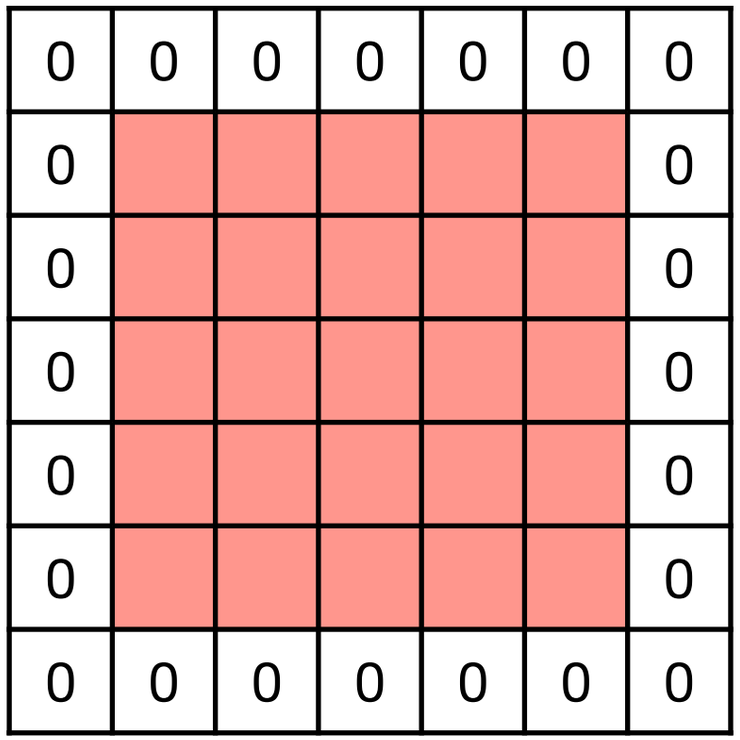
Output

Problem: For large images we need many layers for each output to “see” the whole image image

Want each output unit to combine the information of whole image.
Solution: downsample the input within the network.

Stride

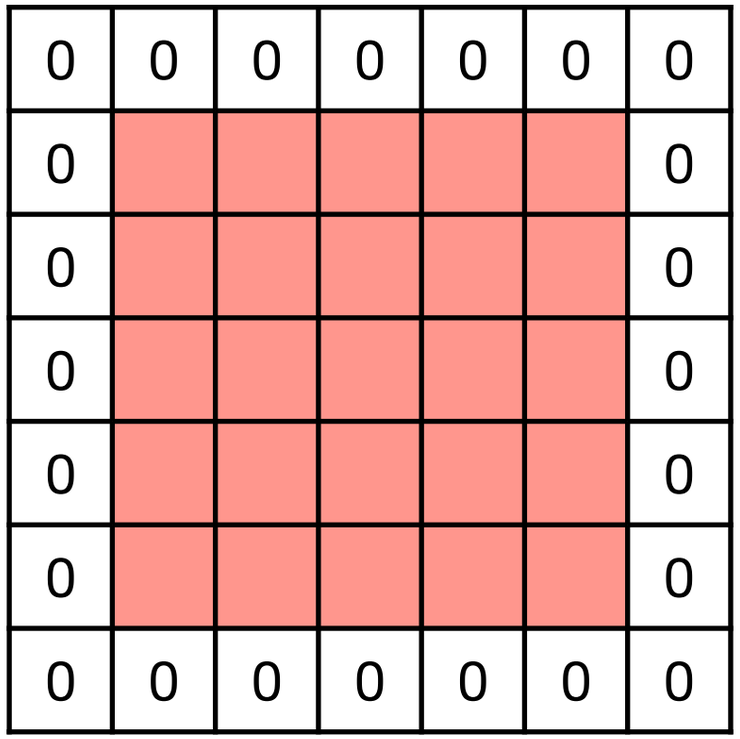
Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

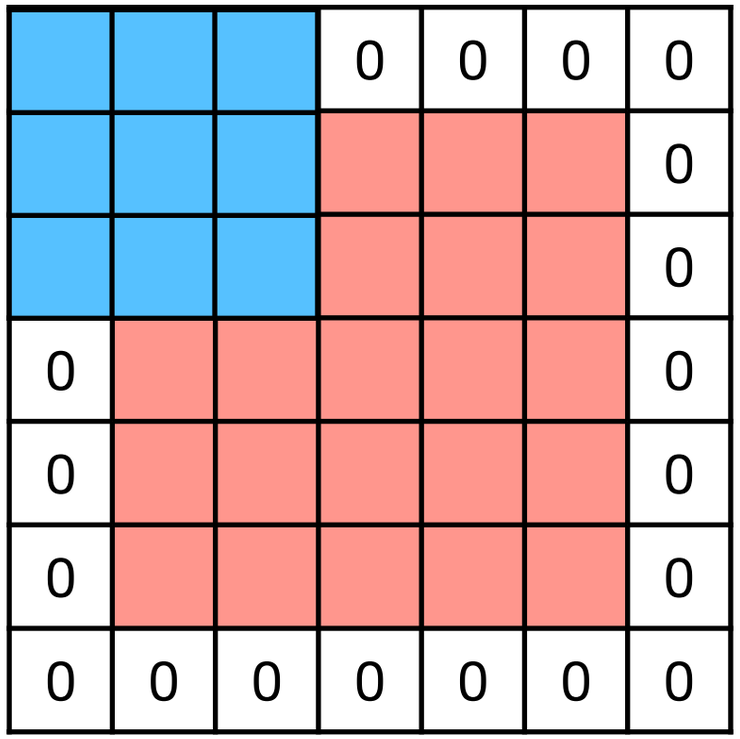
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

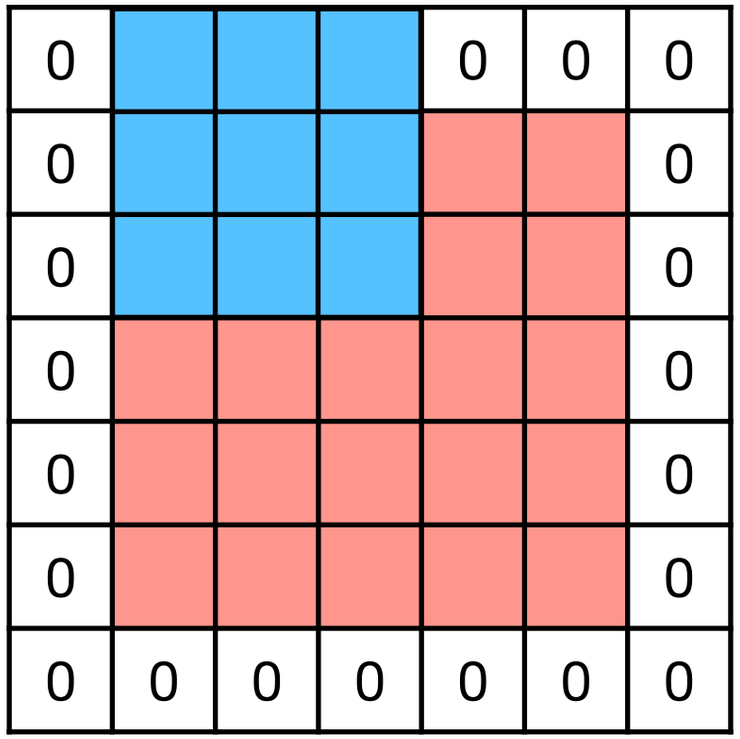
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

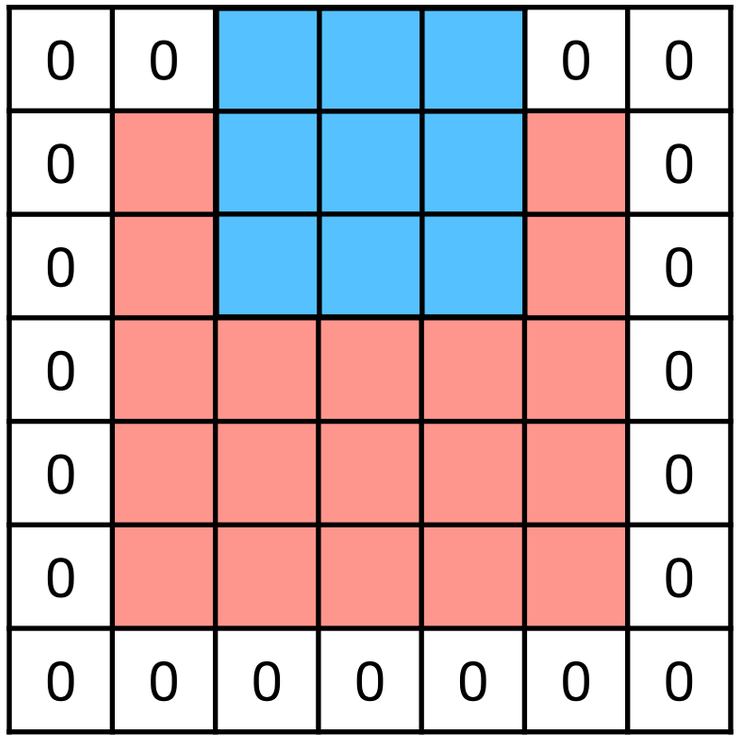
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

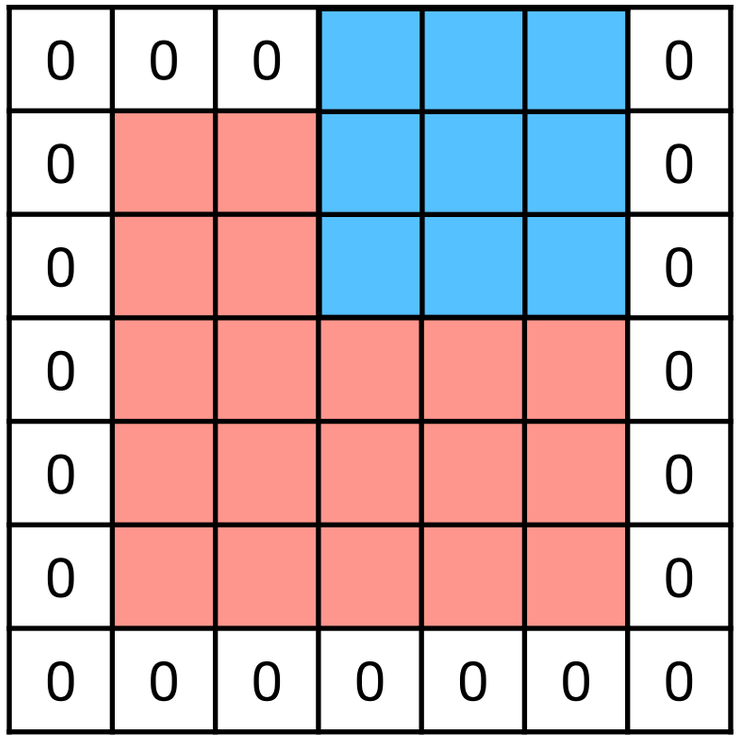
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

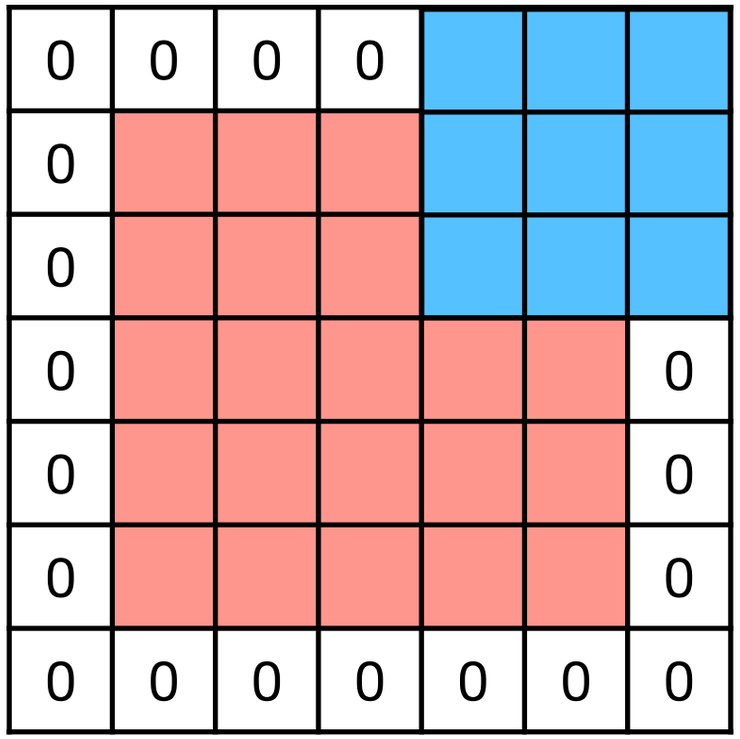
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

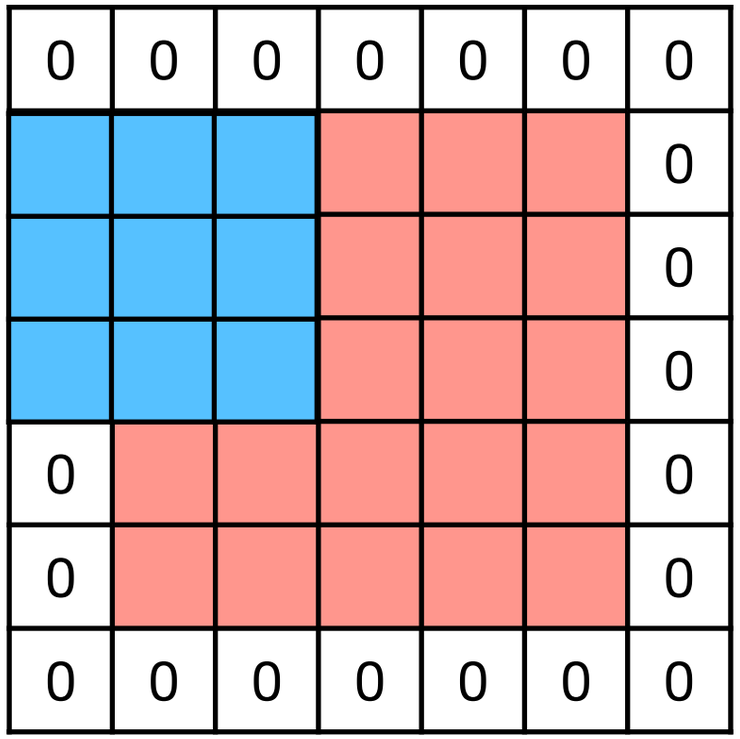
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

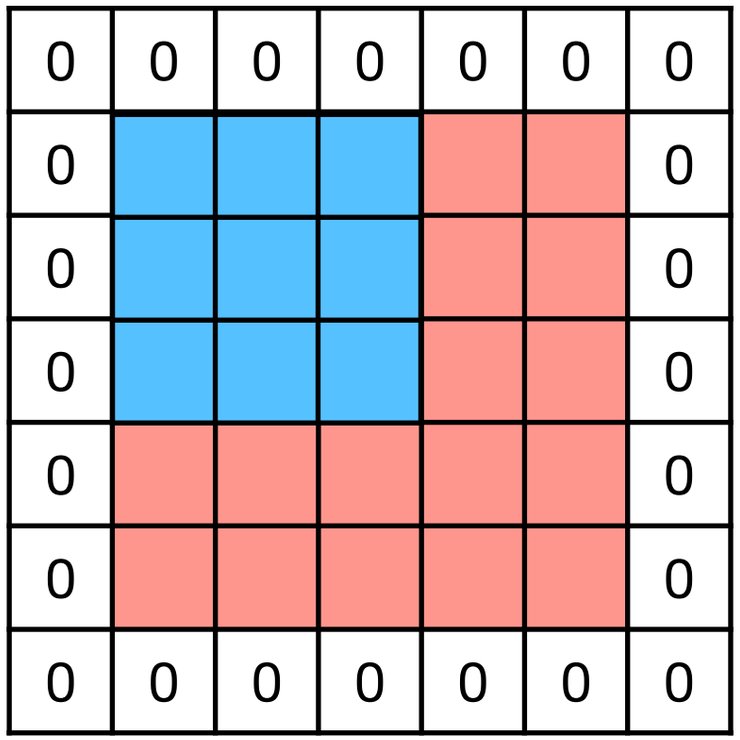
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

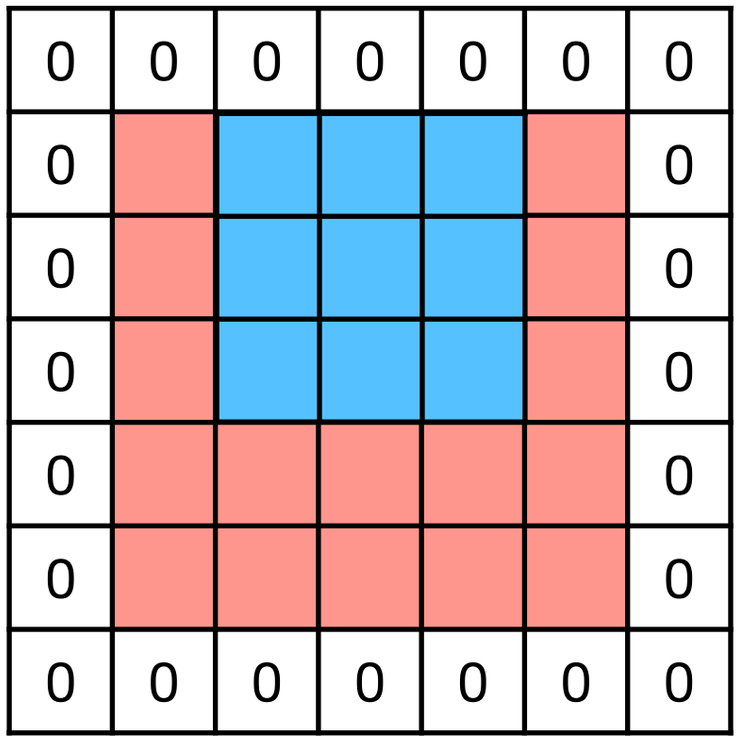
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

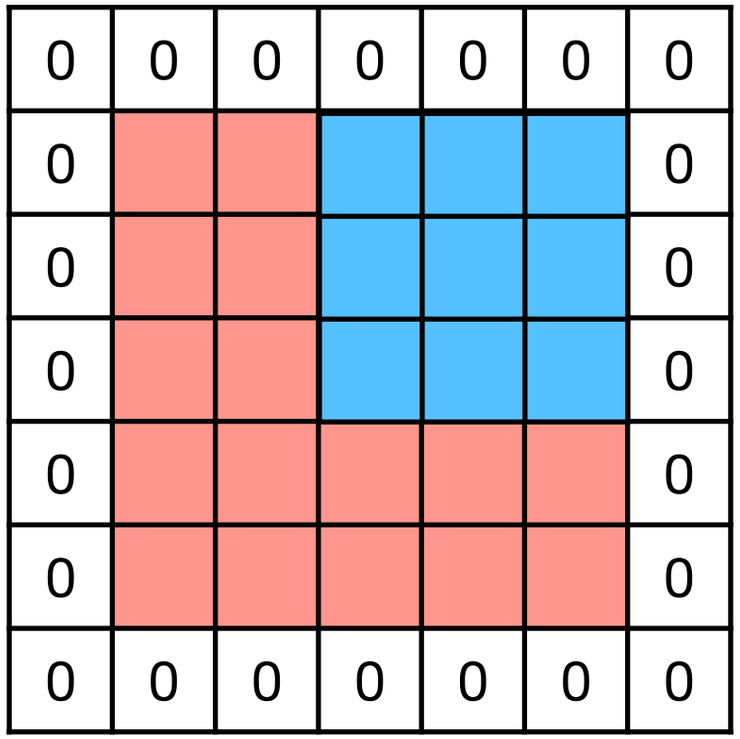
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

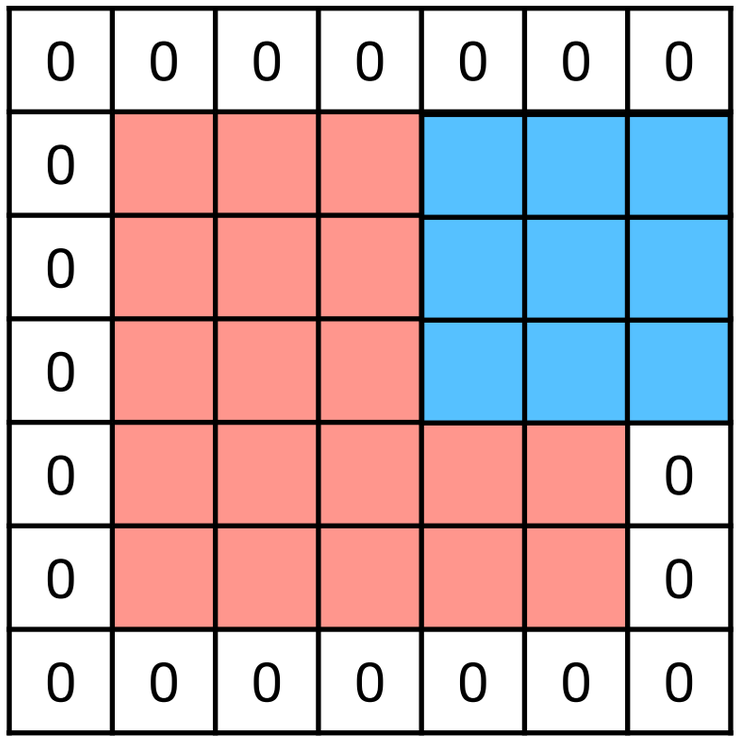
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

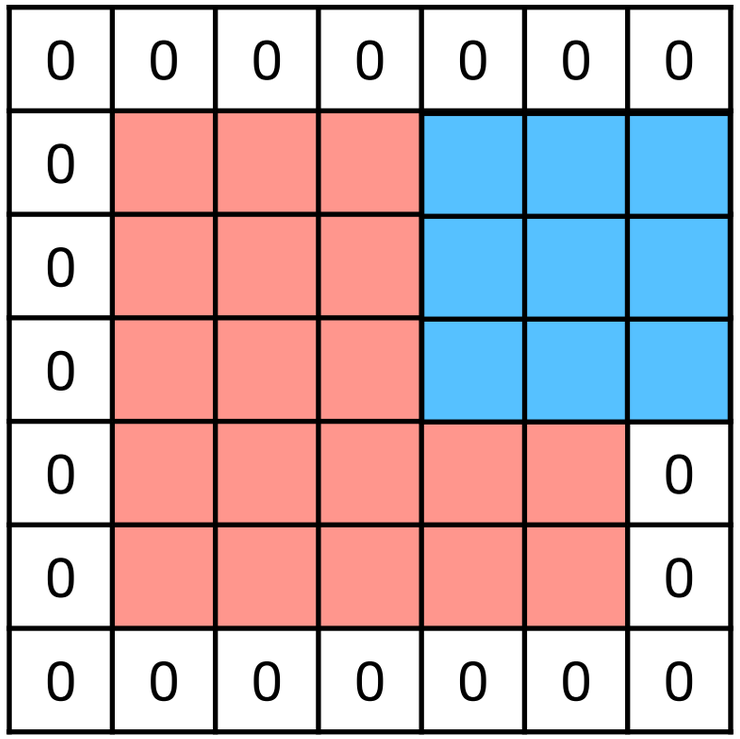
Stride = 1



Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1



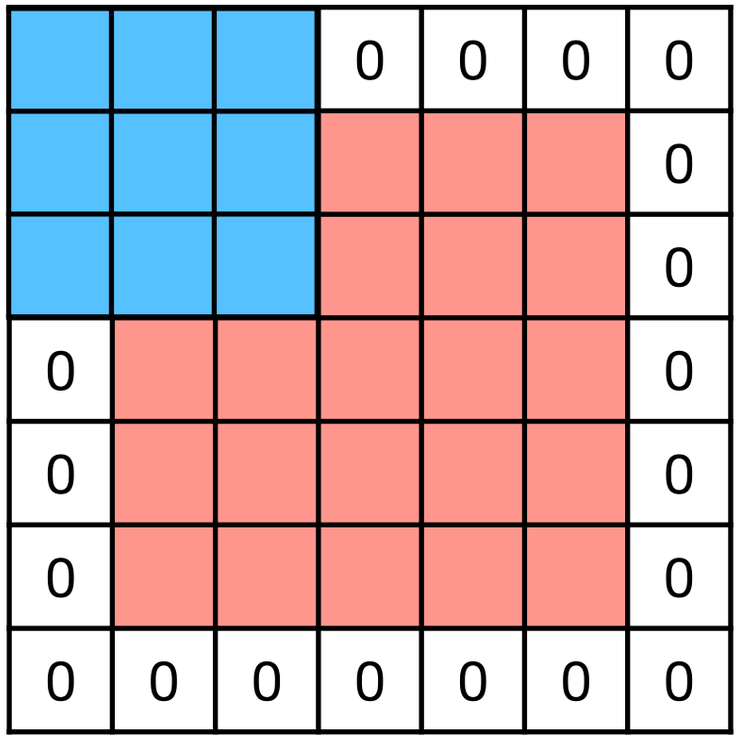
5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1

Stride = 2



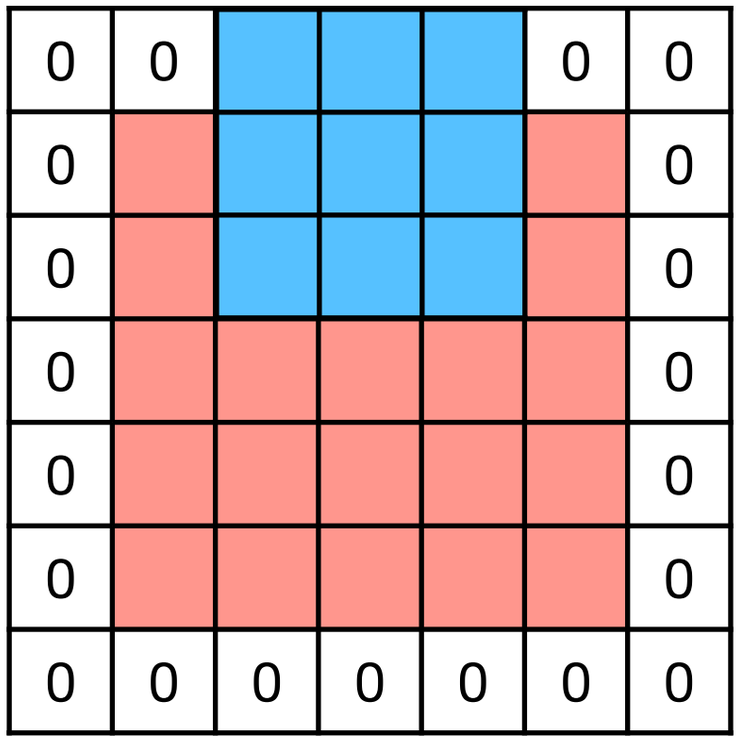
5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1

Stride = 2



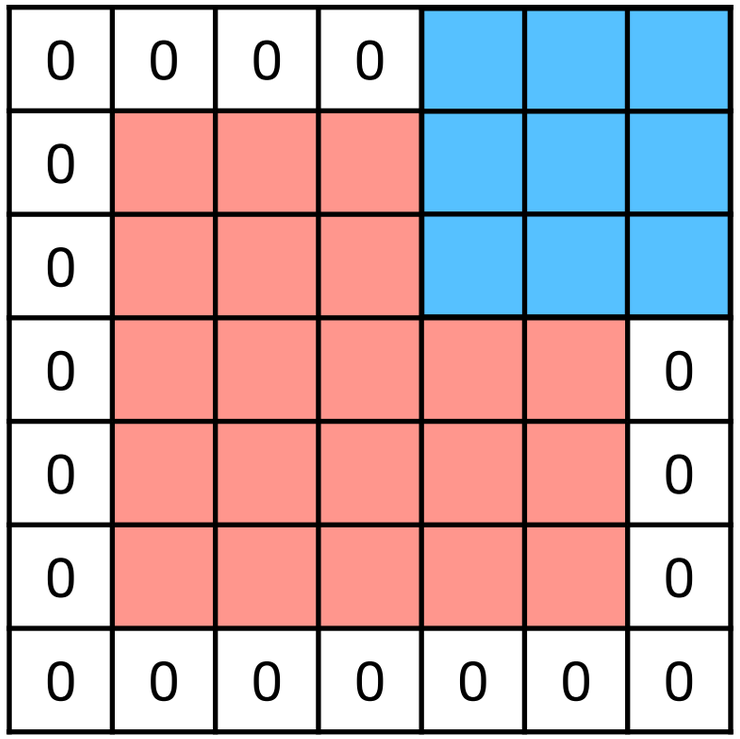
5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1

Stride = 2



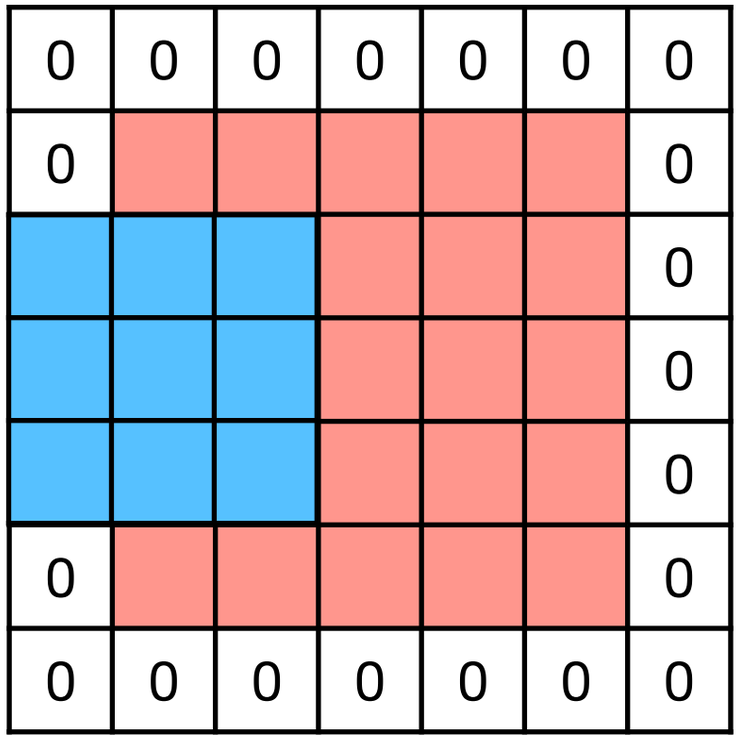
5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1

Stride = 2



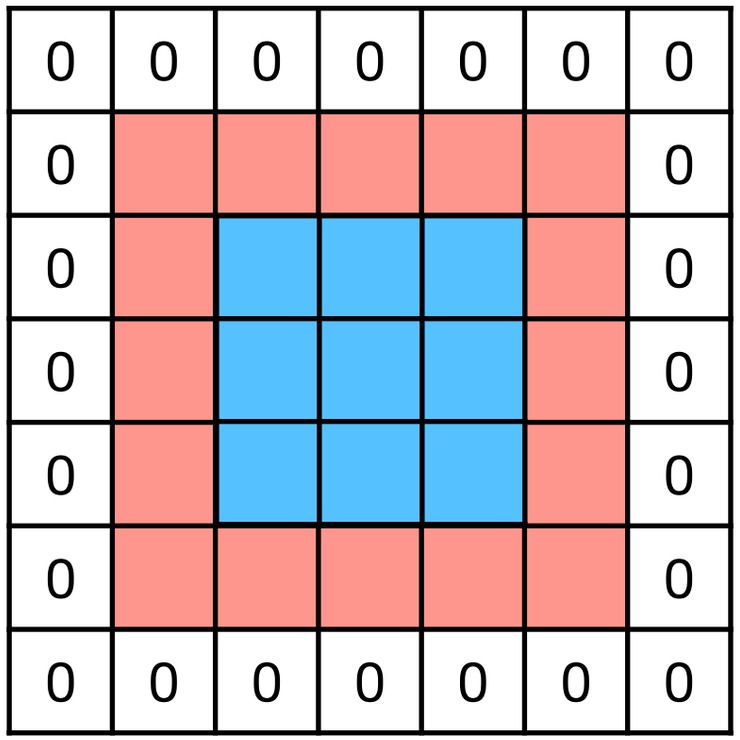
5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1

Stride = 2



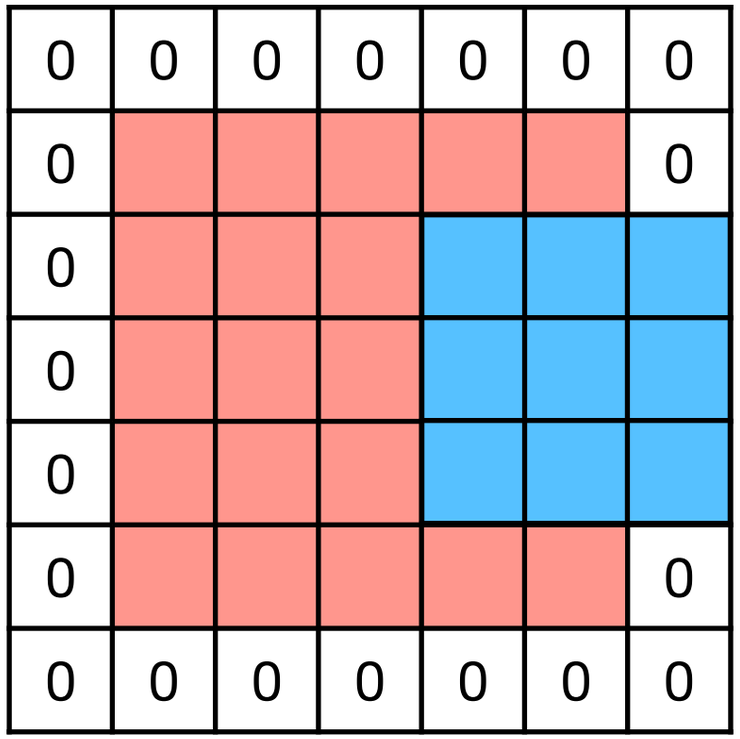
5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

Stride = 1

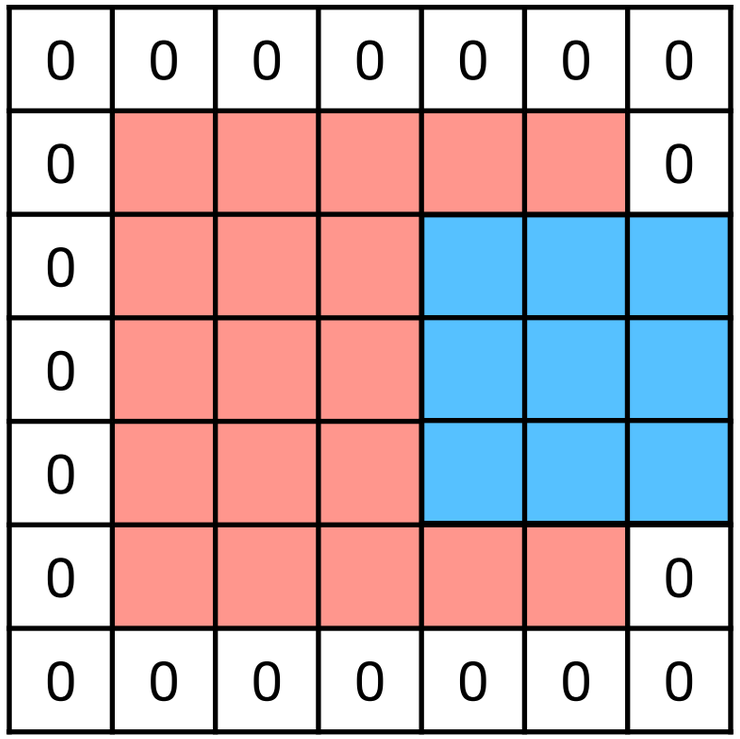
Stride = 2



5*5 to 5*5

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.

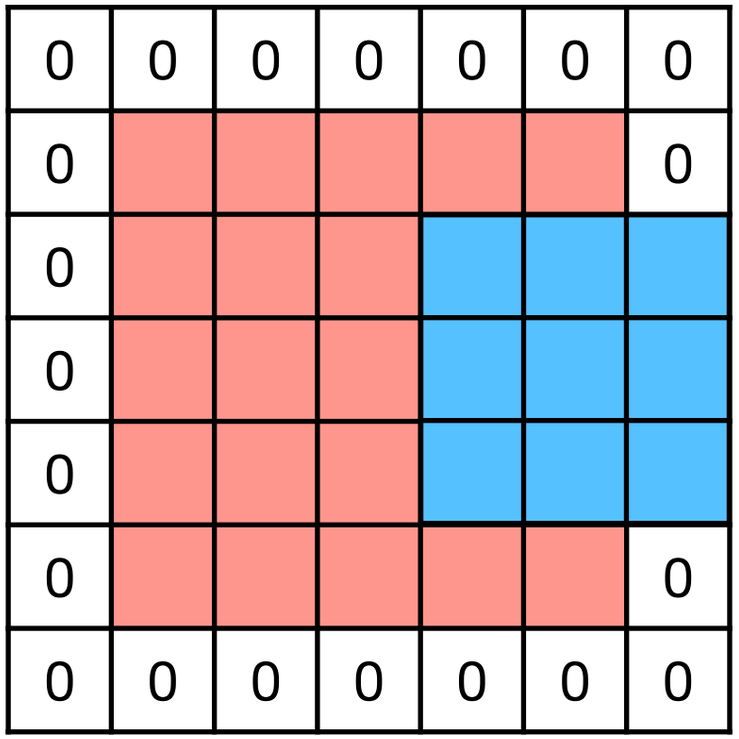


Stride = 1 **5*5 to 5*5**

Stride = 2 **5*5 to 3*3**

Stride

Stride: sliding the filter with a step size.
Large strides indicate sliding with large jumps.



Stride = 1 **5*5 to 5*5**

Stride = 2 **5*5 to 3*3**

Large stride downsamples the input.
Any other way to downsample?

Pooling Layer

Max Pooling

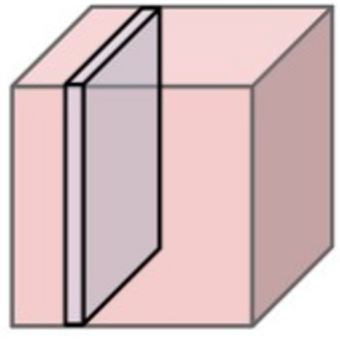
Single depth slice

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

x

y

64 x 224 x 224



Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Introduces **invariance** to small spatial shifts

No learnable parameters!

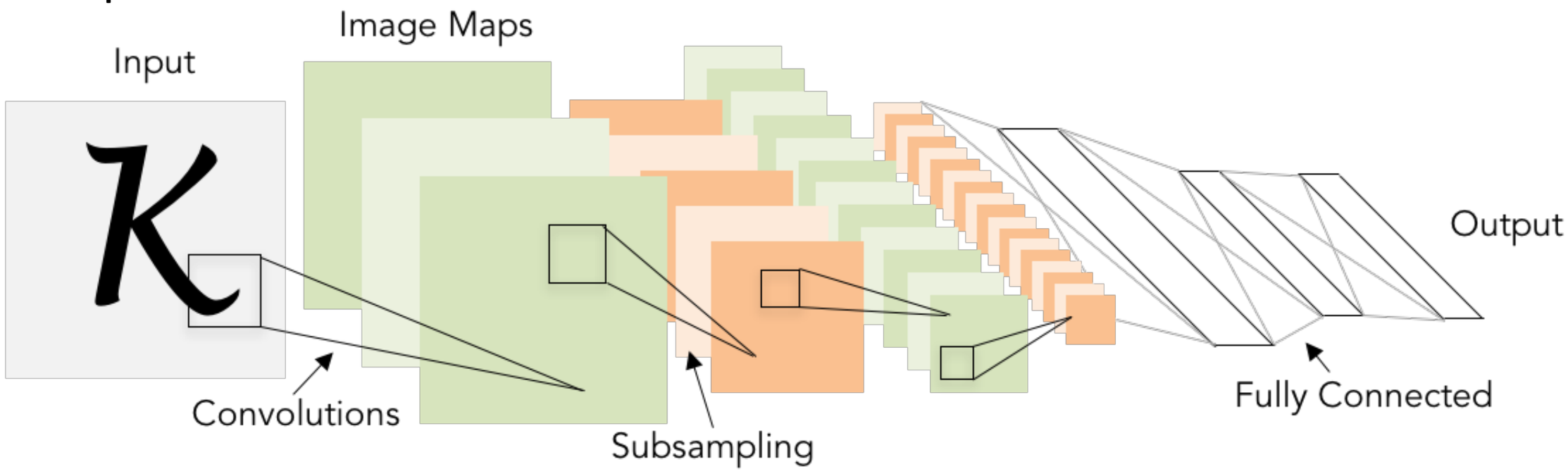
Convolutional Networks

PROC. OF THE IEEE, NOVEMBER 1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Example: LeNet-5



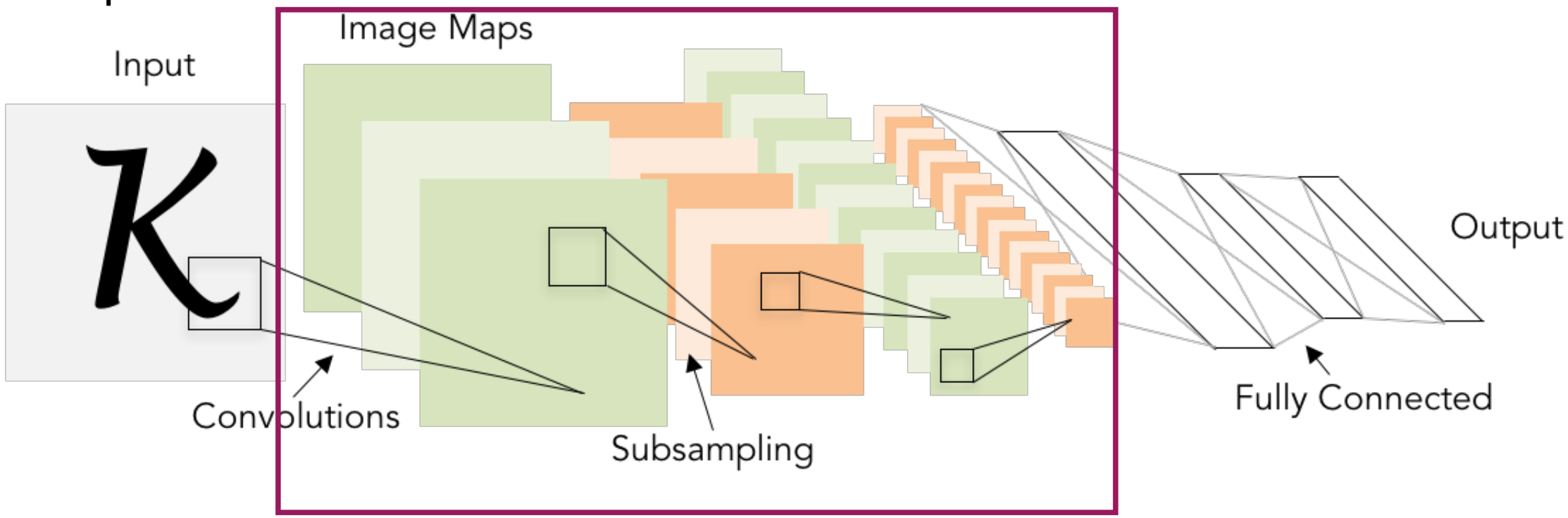
Convolutional Networks

PROC. OF THE IEEE, NOVEMBER 1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Example: LeNet-5



convolutional layers

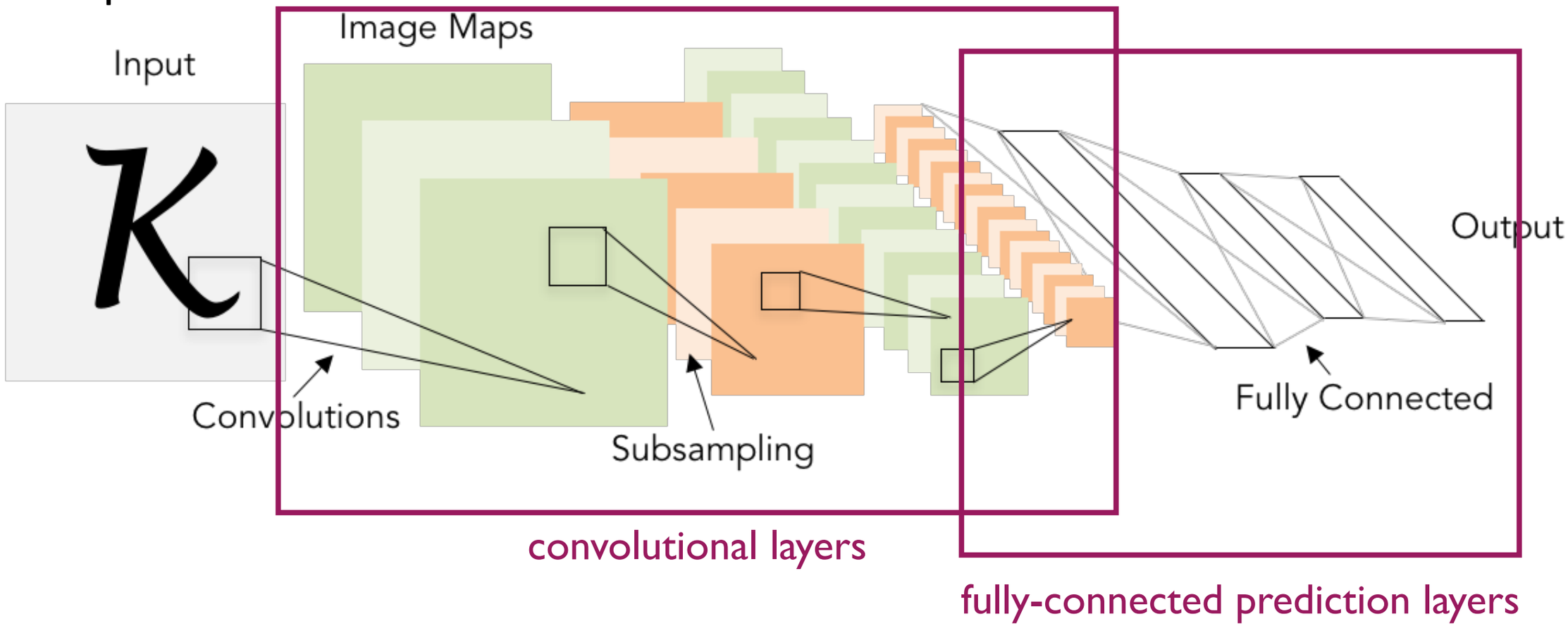
Convolutional Networks

PROC. OF THE IEEE, NOVEMBER 1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

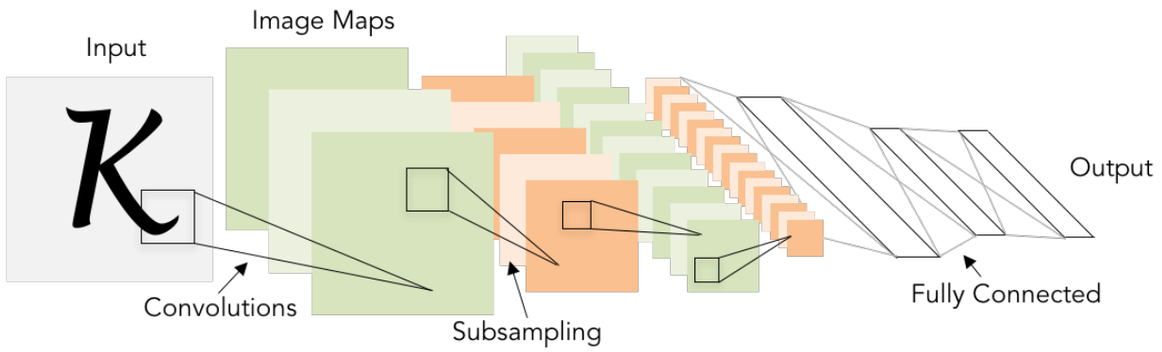
Example: LeNet-5



LeNet-5

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10



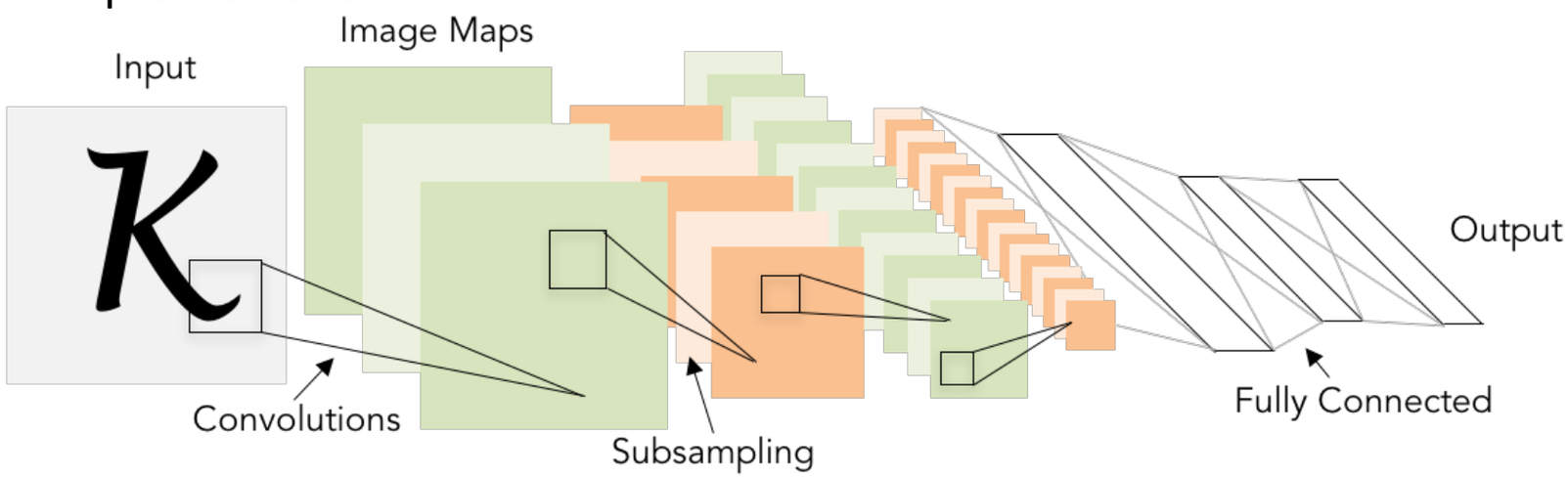
As we go through the network:

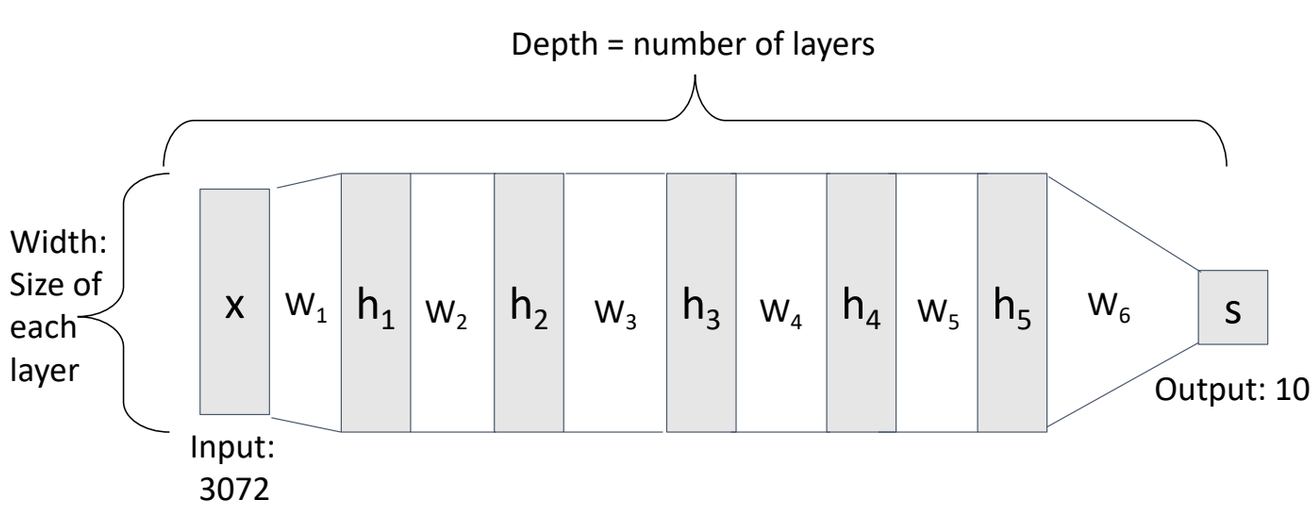
Spatial size **decreases**
(using pooling or strided conv)

Number of channels **increases**
(total “volume” is preserved!)

How Can We Go Deep?

Example: LeNet-5





$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Why not just add more hidden layers?

Machine Learning: IV

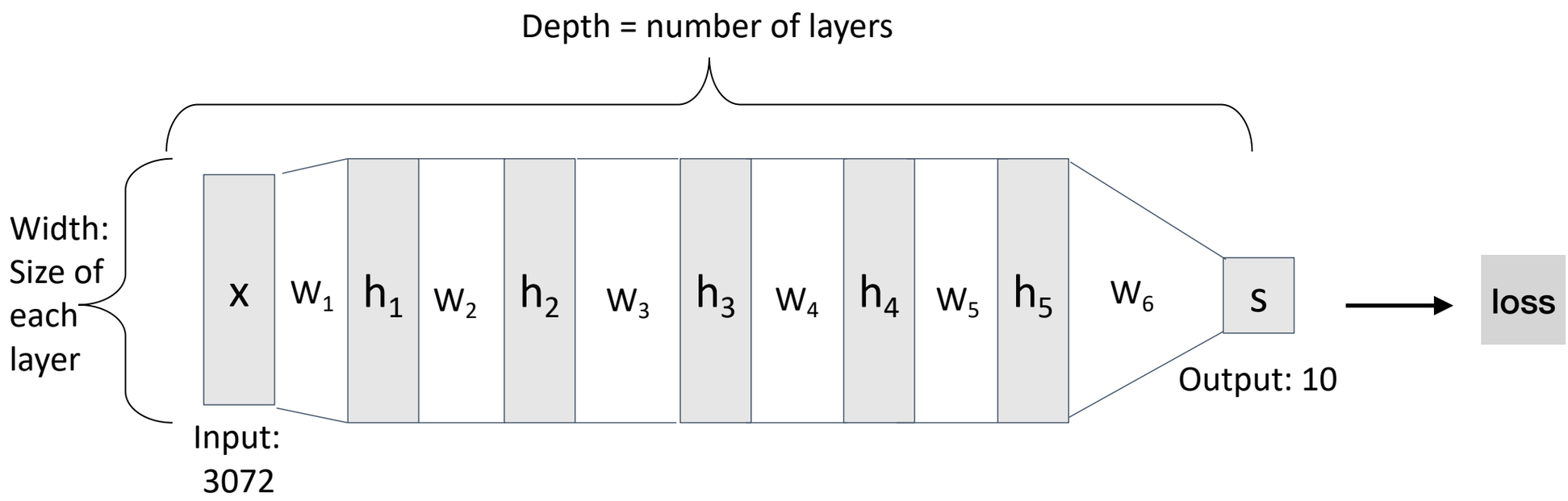
- Convolutional network

- Deep learning
  - Gradient vanishing
  - Deep NN architectures
  - Tricks of the trade

- Take-home messages

Back Propagation & Gradient Vanishing

forward pass



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

backward pass

SGD & BP lies at the heart of NN learning.
In general, for backward pass, the gradient of bottom layers is much smaller than the gradient of top layers.
For deep NN, the gradient is difficult to be passed to the bottom layers.

Gradient Vanishing

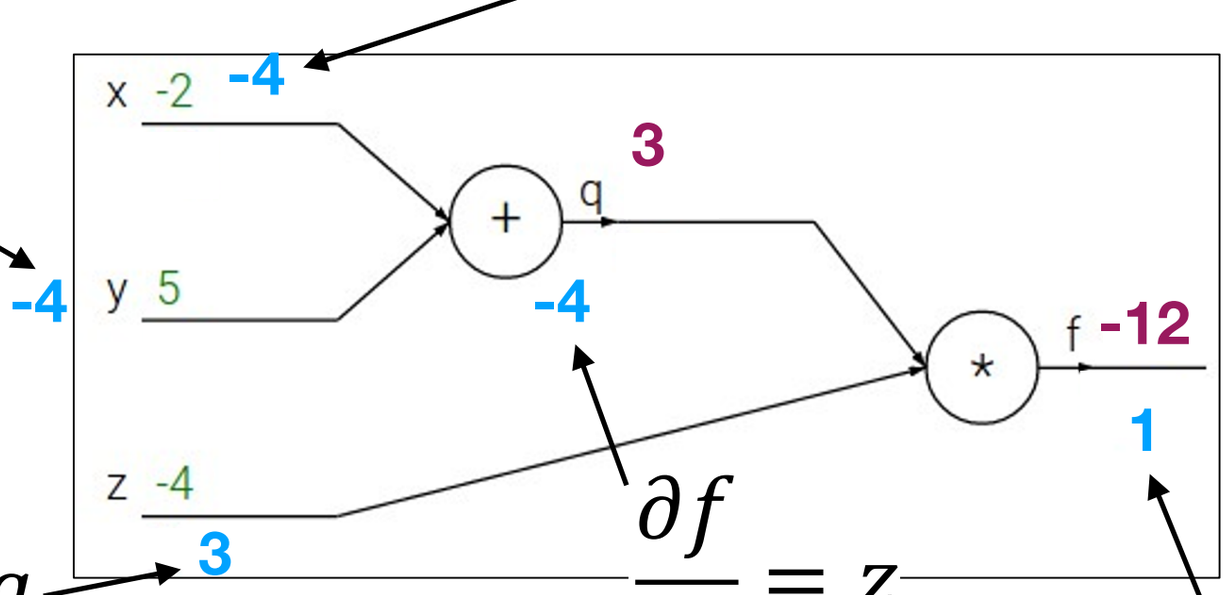
The gradient bottom layers need compound multiplication, leading to gradient vanishing (or explosion).

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$



$$\frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial q} = z$$

$$\frac{\partial f}{\partial f}$$

Deep Learning

A fast learning algorithm for deep belief nets *

Geoffrey E. Hinton and **Simon Osindero**
Department of Computer Science University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4
{hinton, osindero}@cs.toronto.edu

Yee-Whye Teh
Department of Computer Science
National University of Singapore
3 Science Drive 3, Singapore, 117543
tehyw@comp.nus.edu.sg

- In 2006, Hinton proposed a pre-training then BP method to train 4-layer NNs.

- First application in speech recognition.

- Explosion from 2012: ImageNet challenge for image classification.

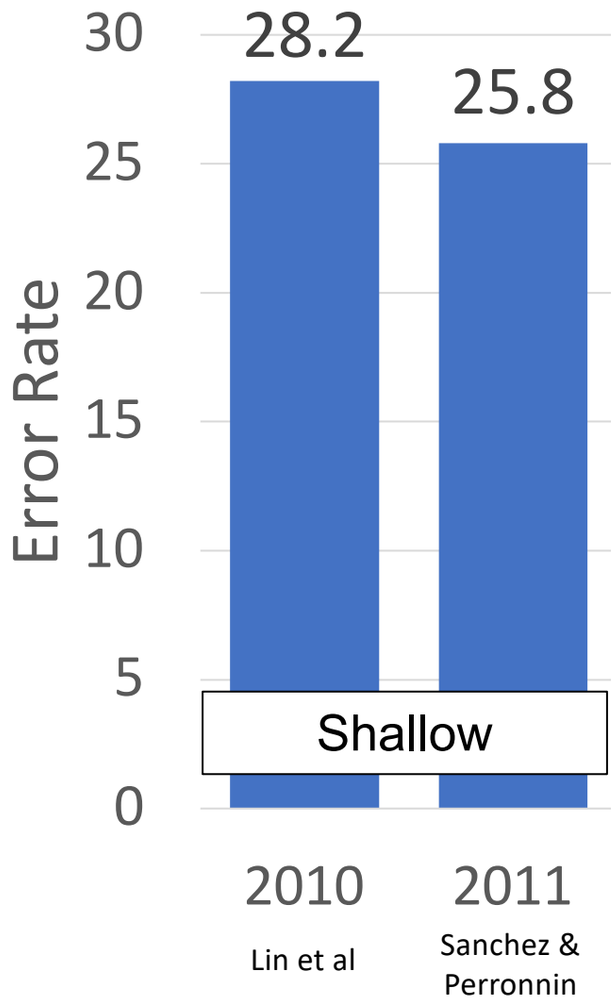
Machine Learning: IV

- Convolutional network

- Deep learning
  - Gradient vanishing
  - Deep NN architectures
    - Tricks of the trade

- Take-home messages

The ImageNet Challenge



- 1000-class image classification.
- Size: height/width about 200 to 300 pix.
- 1.2 million training data.
- 50000 validation data.
- Human error rate: about 5%.

Example: Winner of 2011 ImageNet challenge

Low-level feature extraction ≈ 10k patches per image

- SIFT: 128-dim
- color: 96-dim

} reduced to 64-dim with PCA

FV extraction and compression:

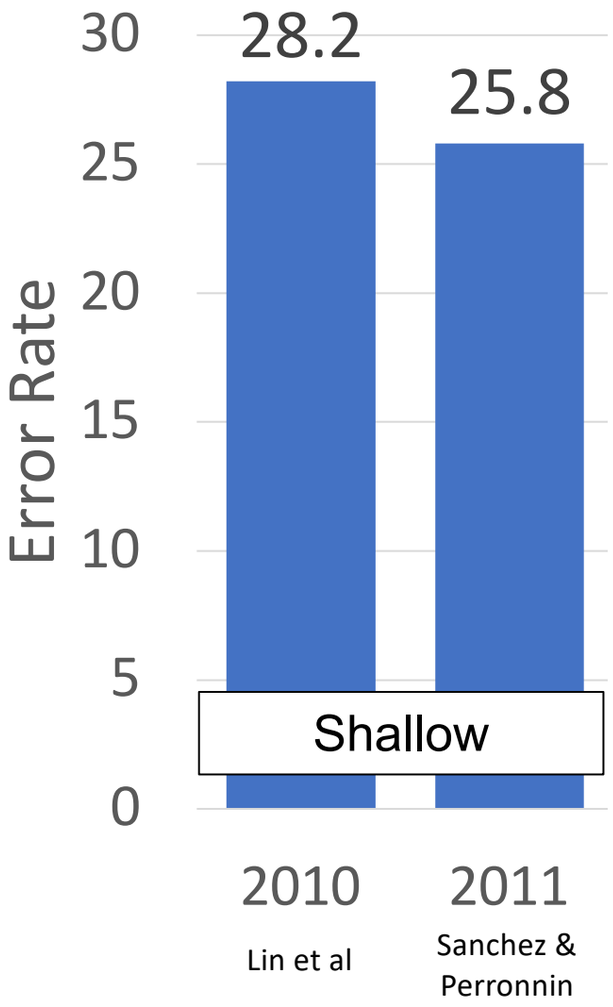
- N=1,024 Gaussians, R=4 regions ⇨ 520K dim x 2
- compression: G=8, b=1 bit per dimension

One-vs-all SVM learning with SGD

Late fusion of SIFT and color systems

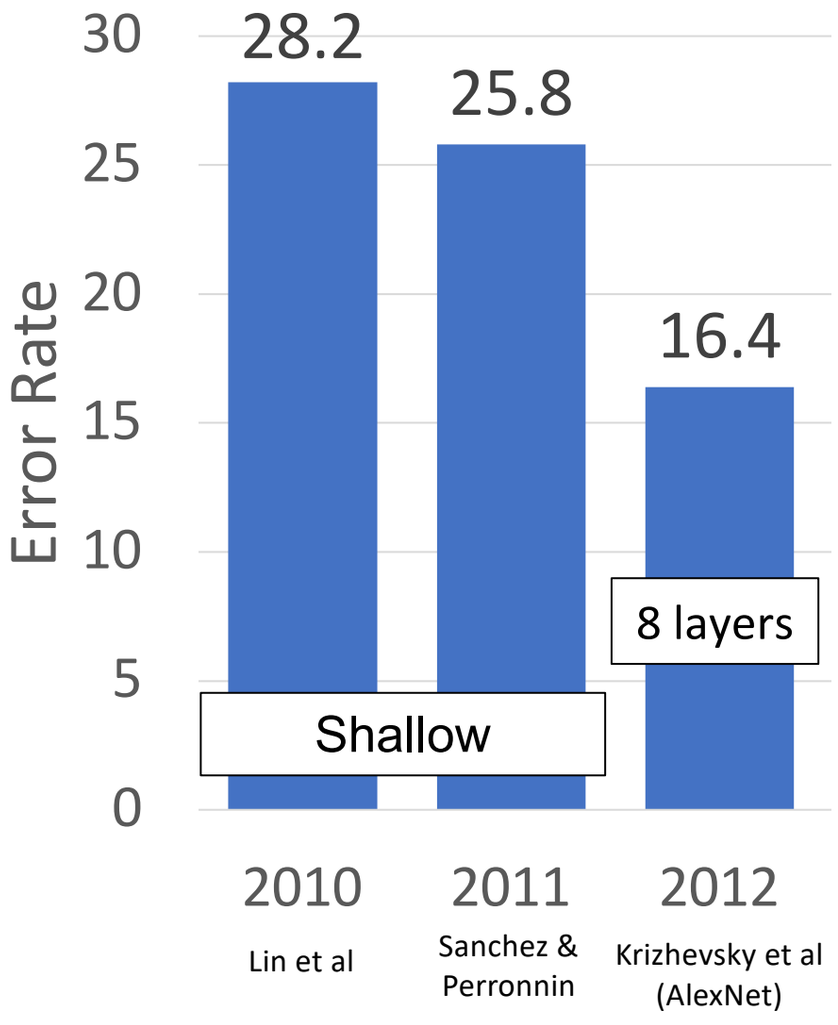
F. Perronnin, J. Sánchez, "Compressed Fisher vectors for LSVRC", PASCAL VOC / ImageNet workshop, ICCV, 2011.

The ImageNet Challenge



- 1000-class image classification.
- Size: height/width about 200 to 300 pix.
- 1.2 million training data.
- 50000 validation data.
- Human error rate: about 5%.

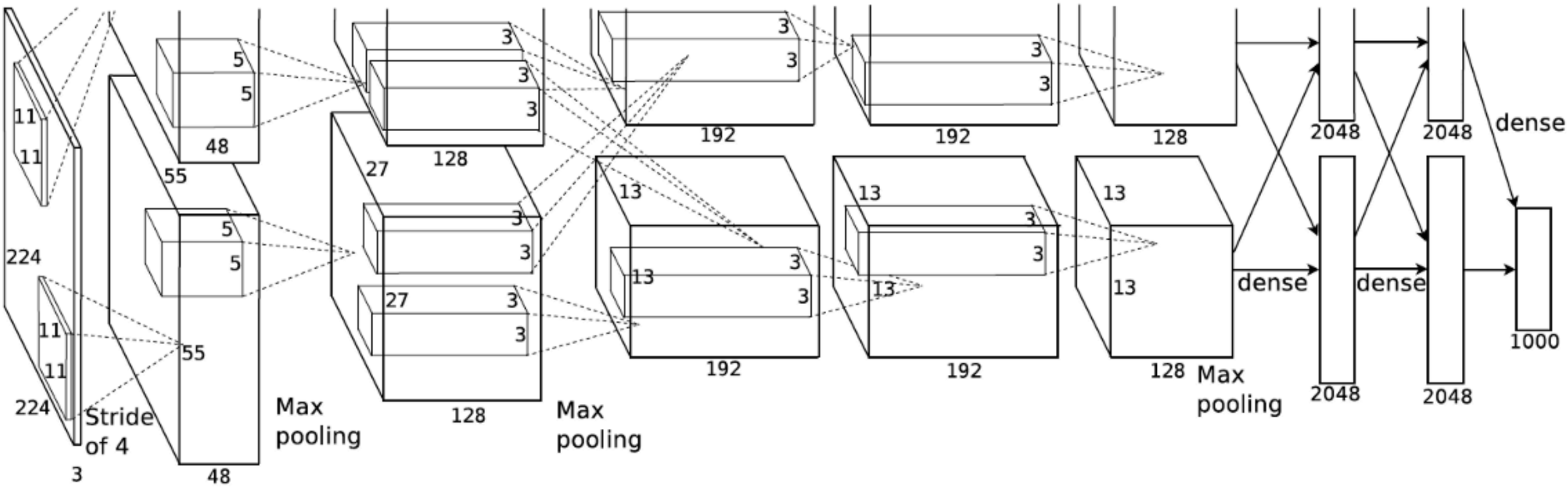
The ImageNet Challenge



- 1000-class image classification.
- Size: height/width about 200 to 300 pix.
- 1.2 million training data.
- 50000 validation data.
- Human error rate: about 5%.

In 2012, deep learning (Hinton group) achieves a big breakthrough over shallow methods.

AlexNet



227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

Used “Local response normalization”; Not used anymore

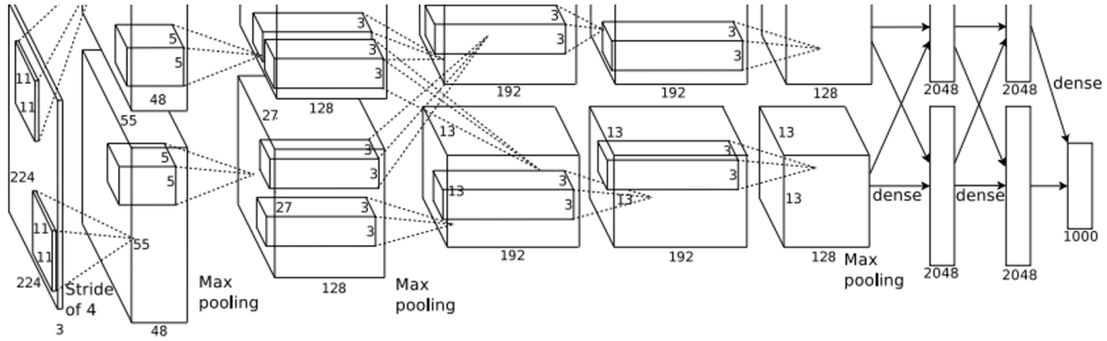
Trained on two GTX 580 GPUs – only 3GB of memory each! Model split over two GPUs

AlexNet

AlexNet

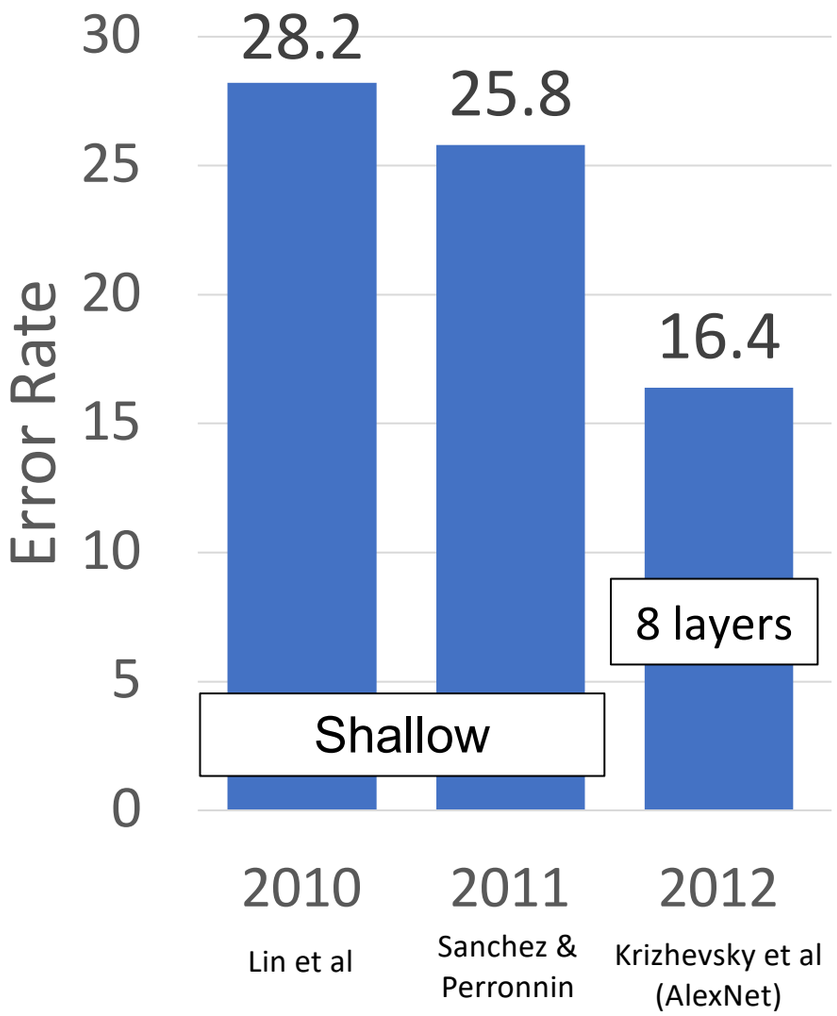
How to choose this?

Trial and error =(

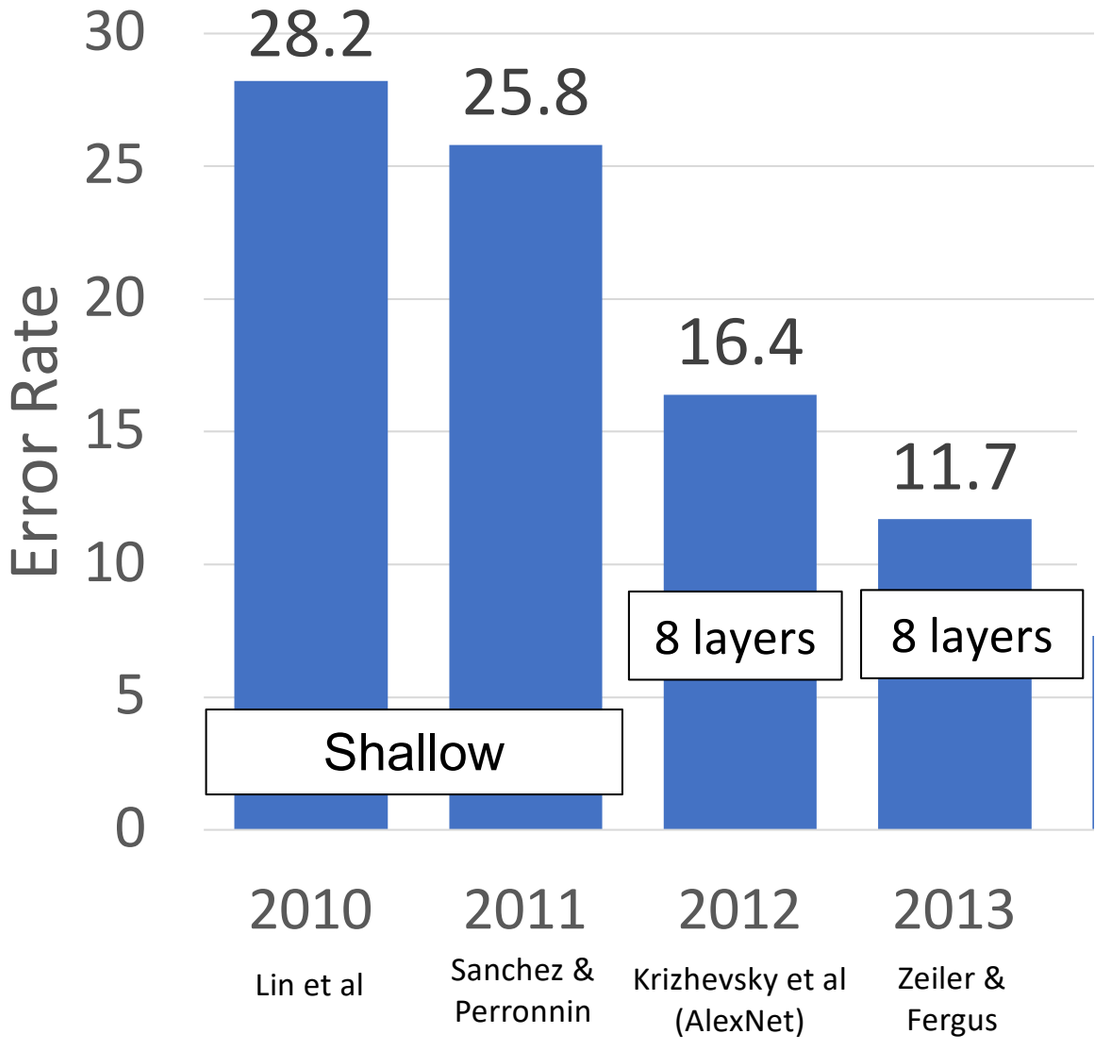


	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

The ImageNet Challenge



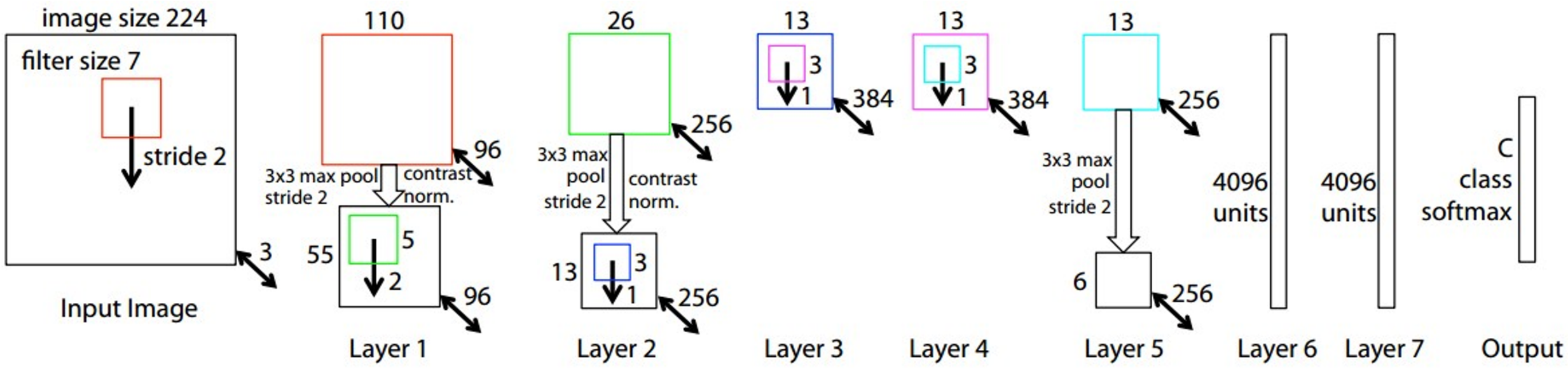
The ImageNet Challenge



ZFNet

ZFNet: A Bigger AlexNet

ImageNet top 5 error: 16.4% -> 11.7%



AlexNet but:

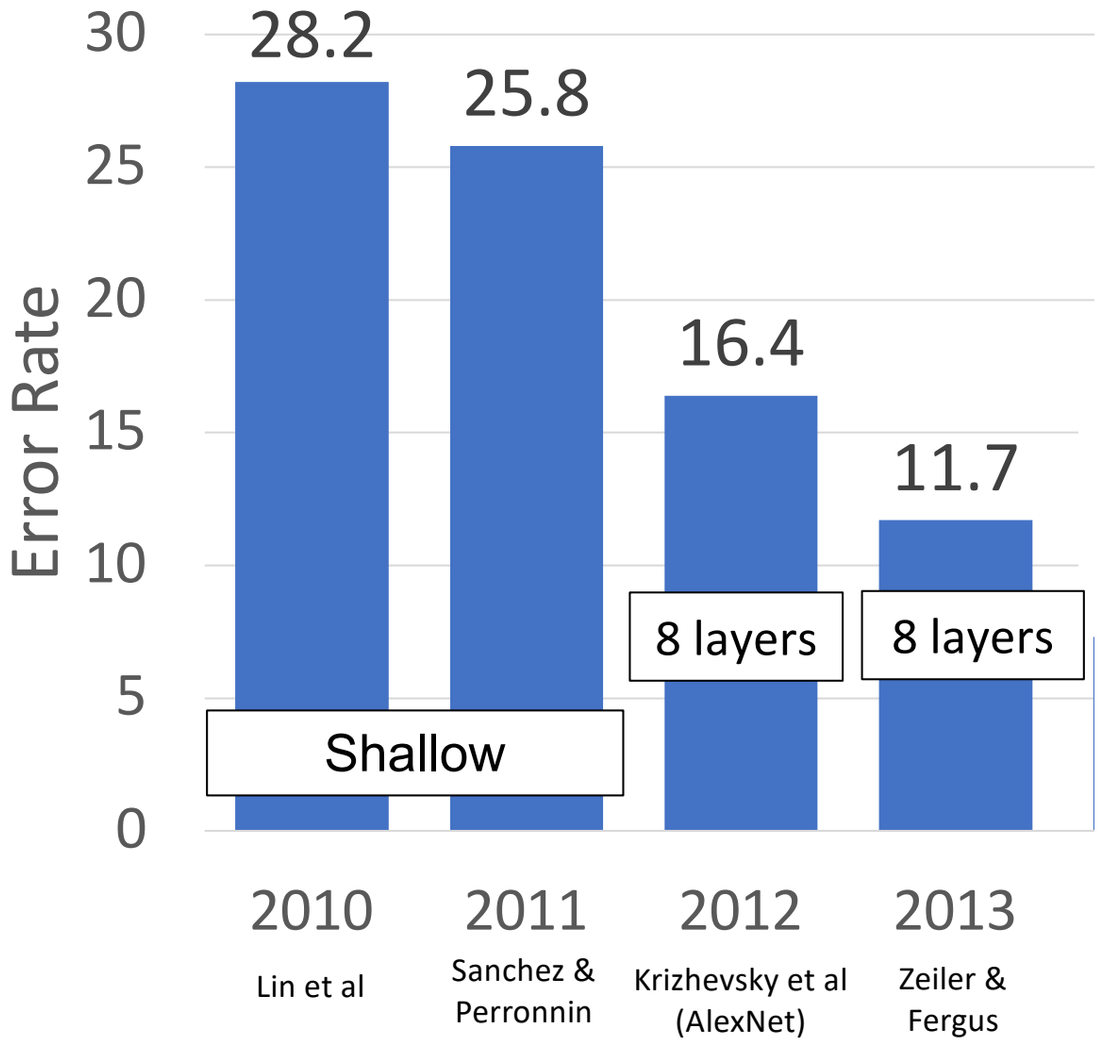
CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

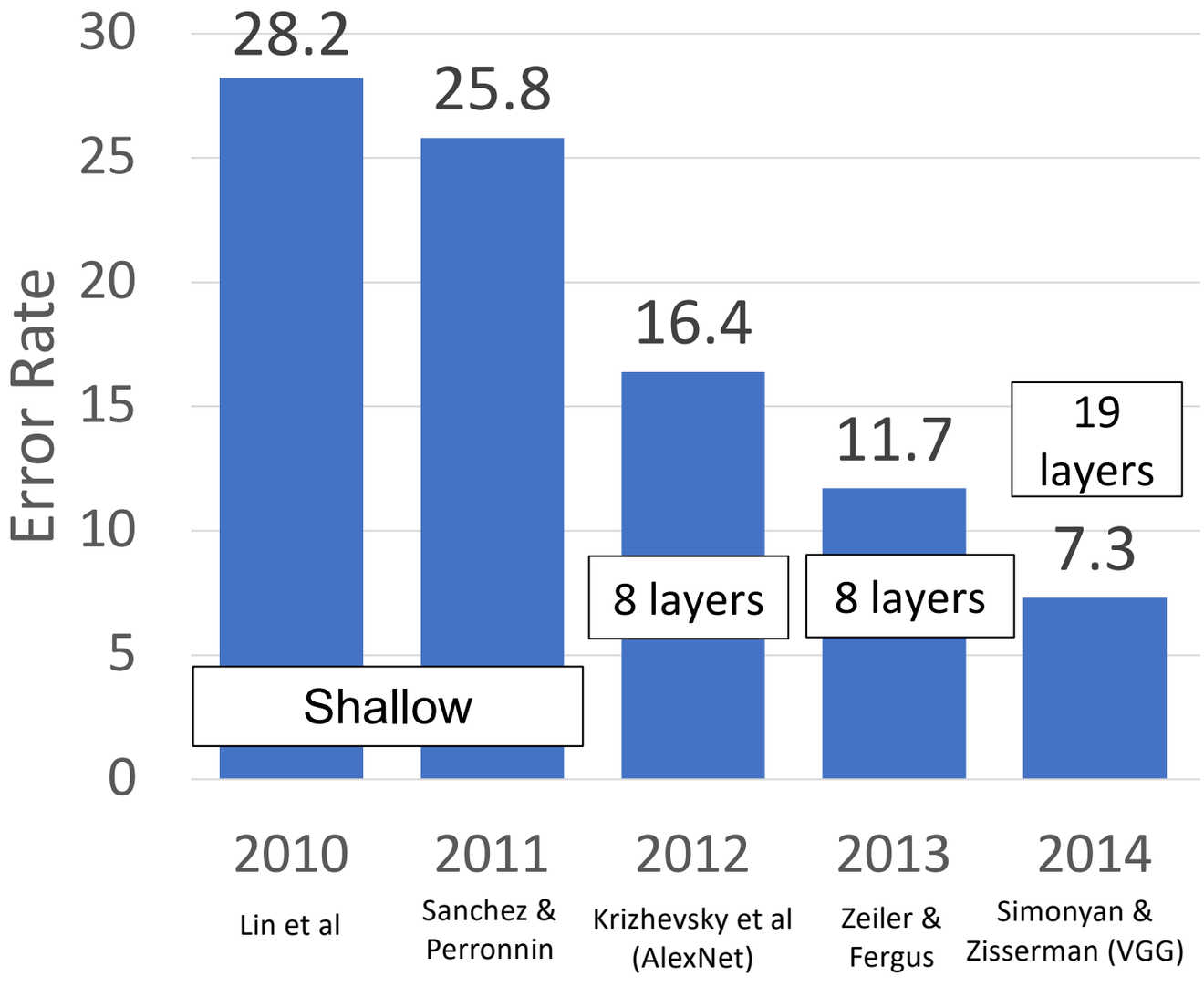
More trial and error =(

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

The ImageNet Challenge



The ImageNet Challenge



VGG Net

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

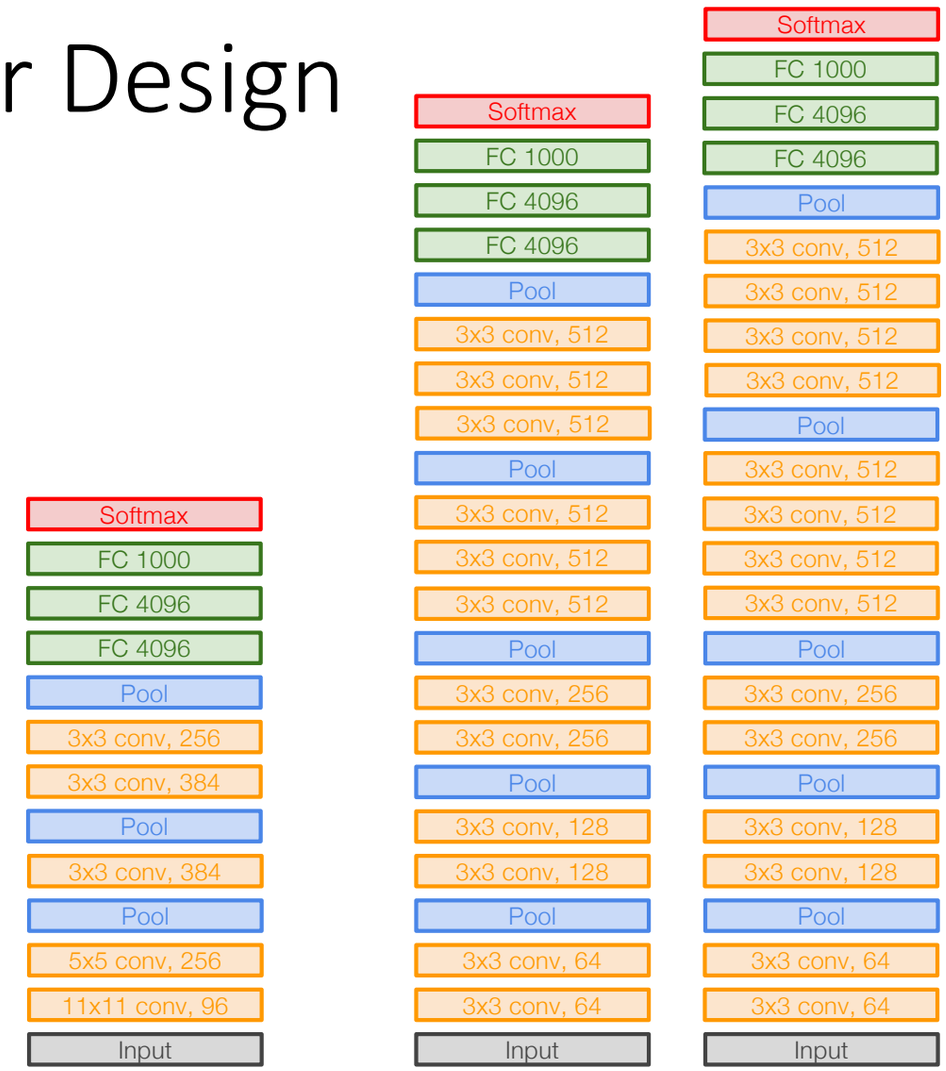
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

Stage 4: conv-conv-conv-[conv]-pool

Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5)



Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

VGG Net

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

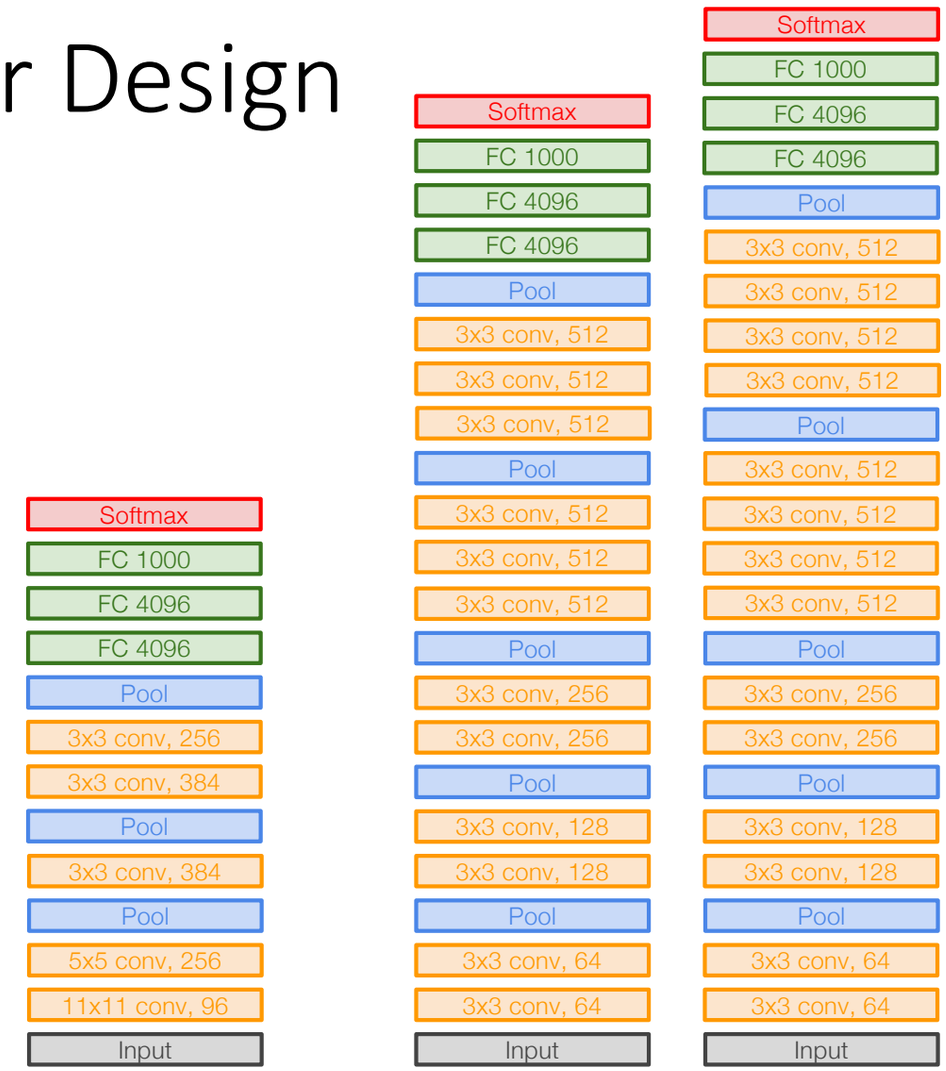
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

Stage 4: conv-conv-conv-[conv]-pool

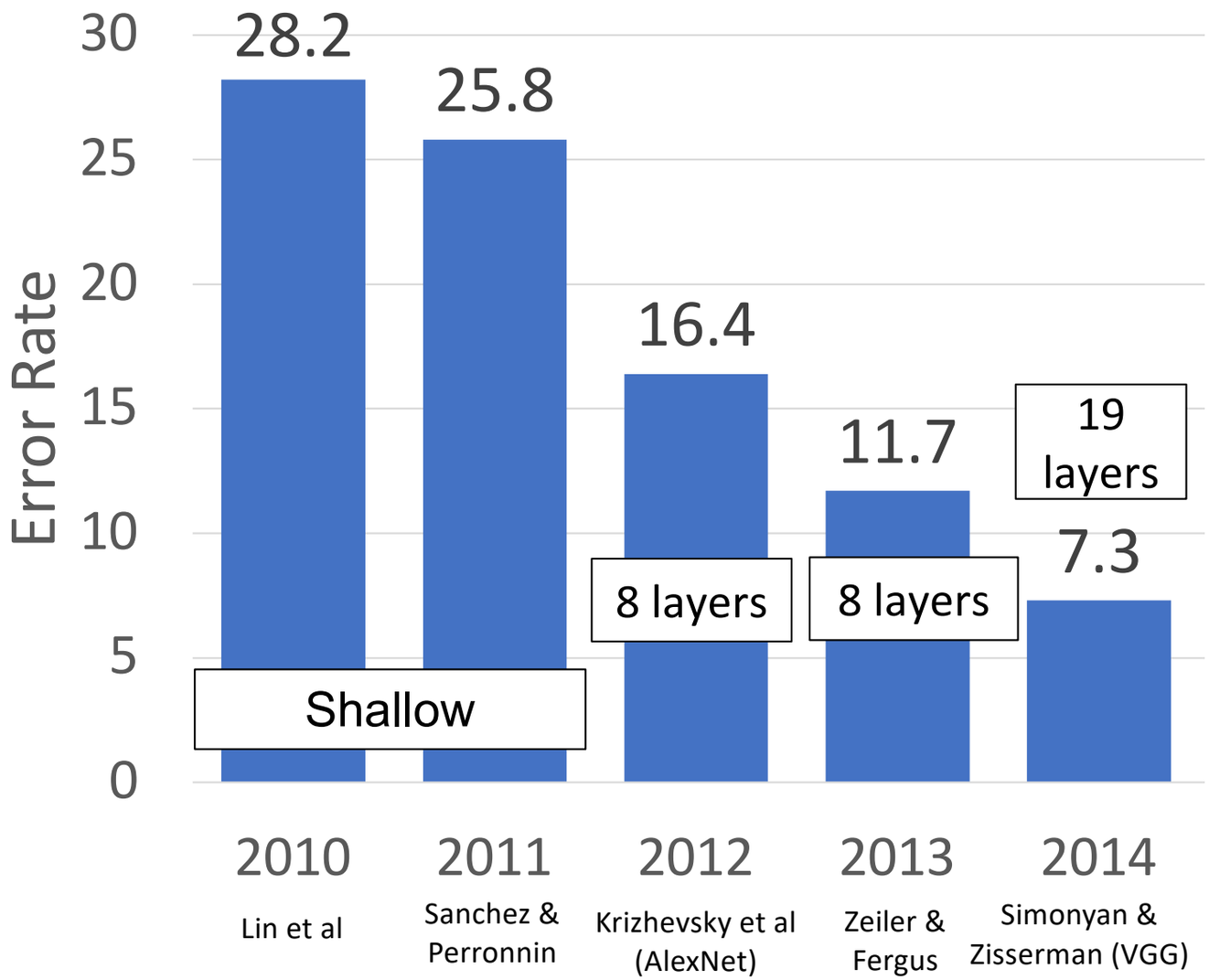
Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5)

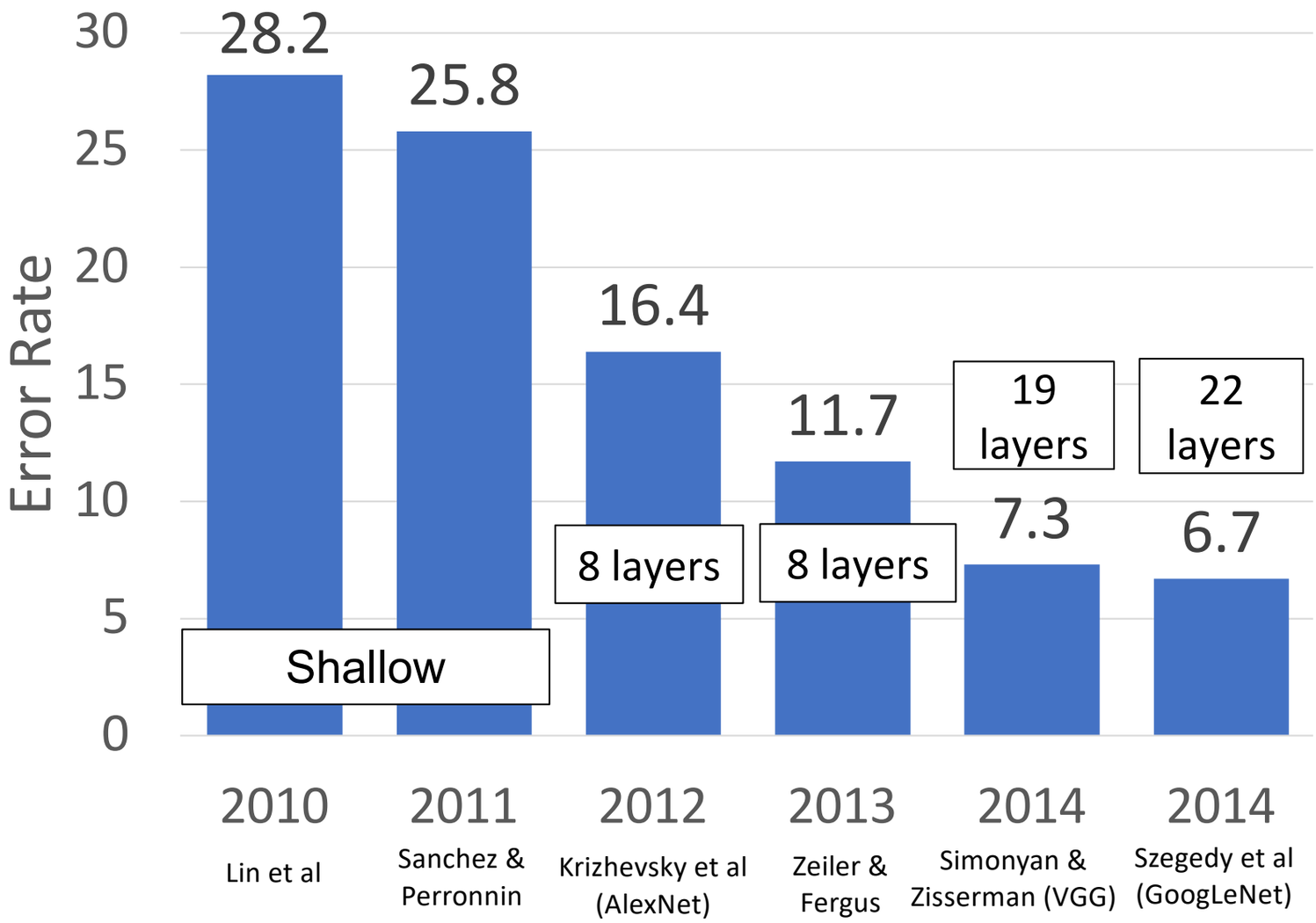


Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

The ImageNet Challenge



The ImageNet Challenge

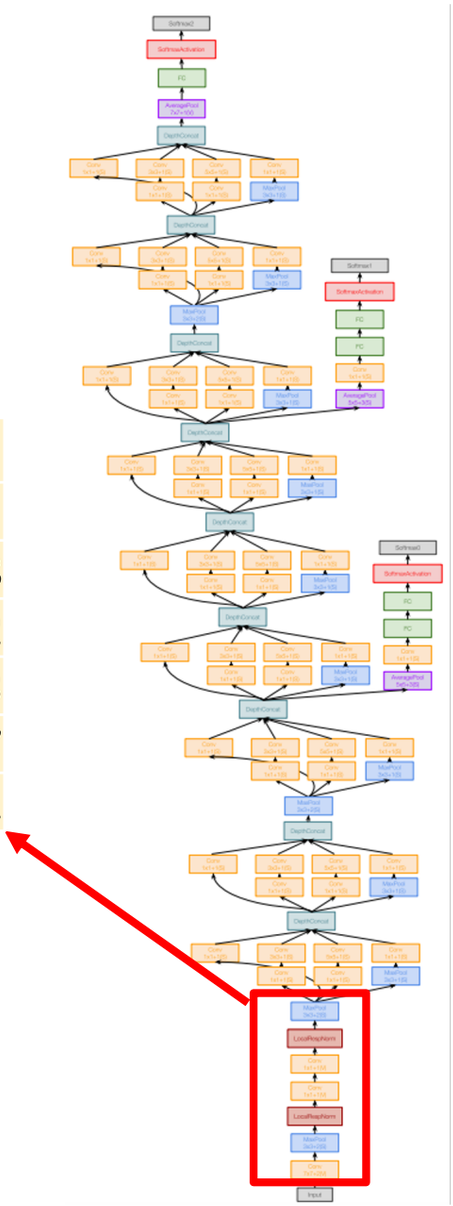


GoogLeNet

GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

	Input size			Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1



Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

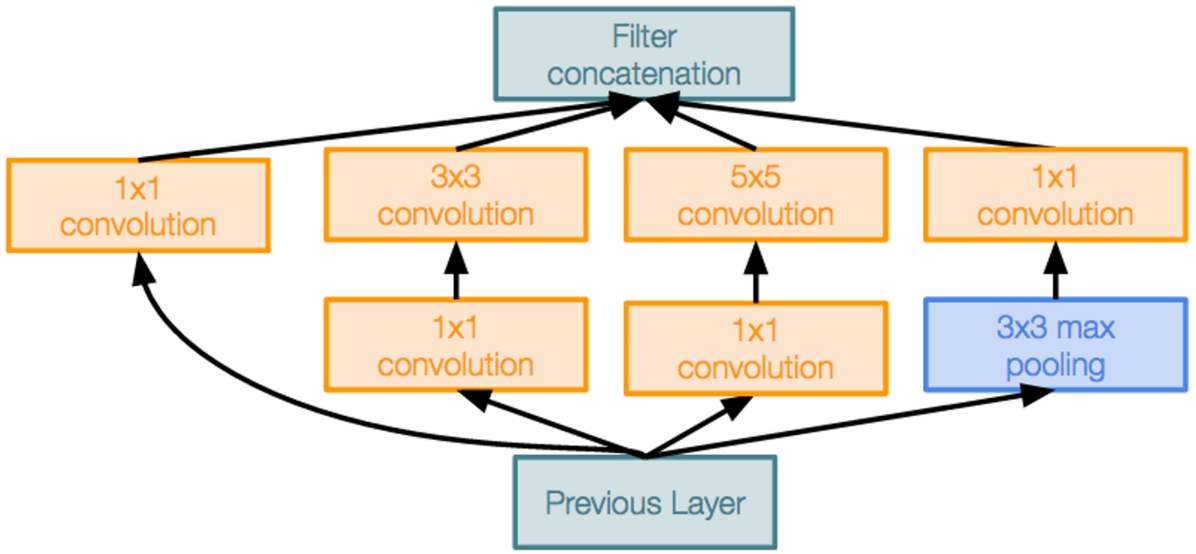
Szegedy et al, “Going deeper with convolutions”, CVPR 2015

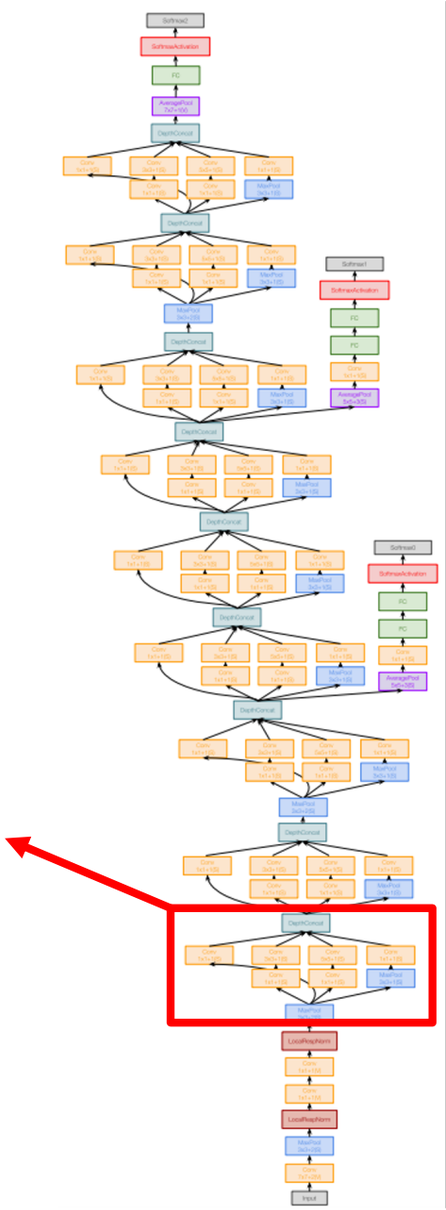
GoogLeNet

GoogLeNet: Inception Module

Inception module
Local unit with
parallel branches

Local structure repeated
many times throughout the
network



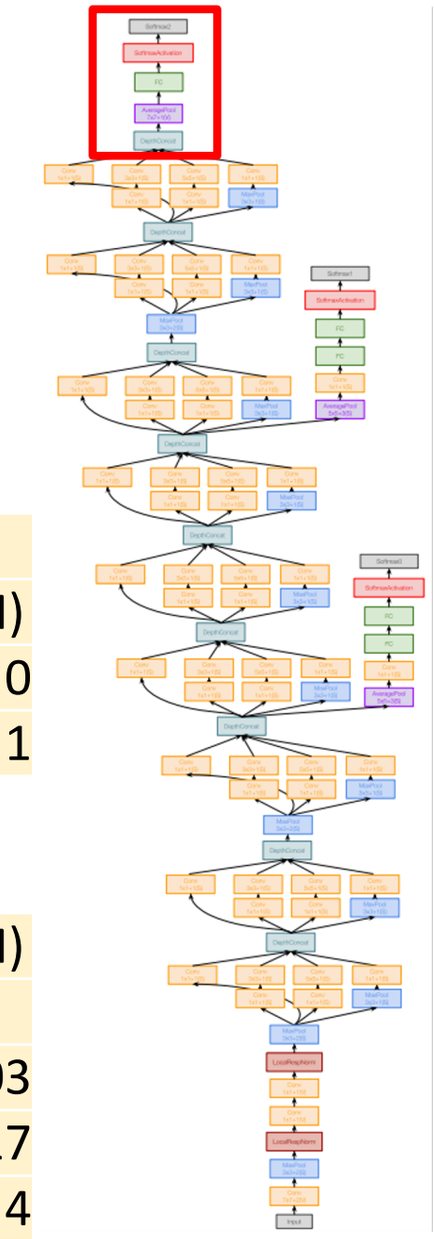


Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet

GoogLeNet: Global Average Pooling

No large FC layers at the end! Instead uses "global average pooling" to collapse spatial dimensions, and one linear layer to produce class scores (Recall VGG-16: Most parameters were in the FC layers!)



	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

Compare with VGG-16:

Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4

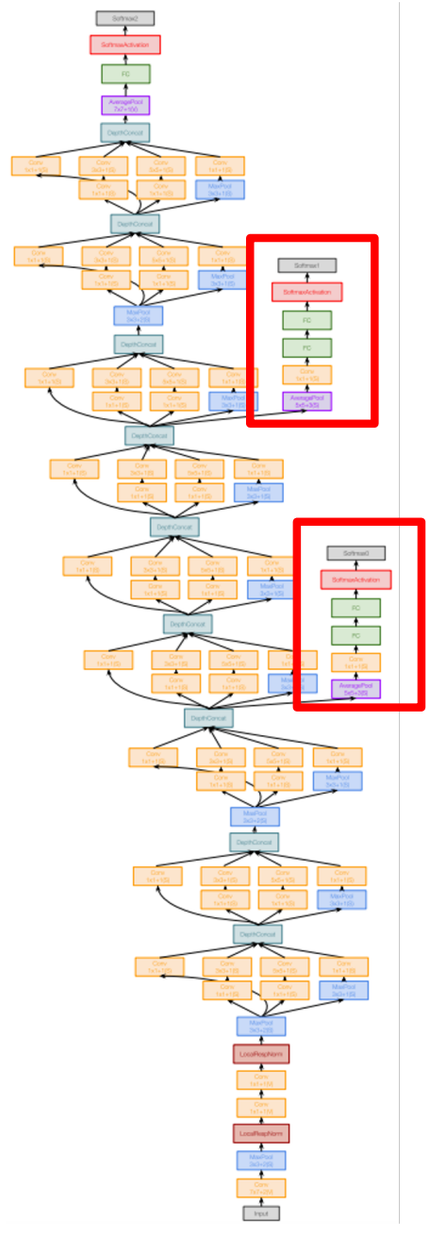
GoogLeNet

GoogLeNet: Auxiliary Classifiers

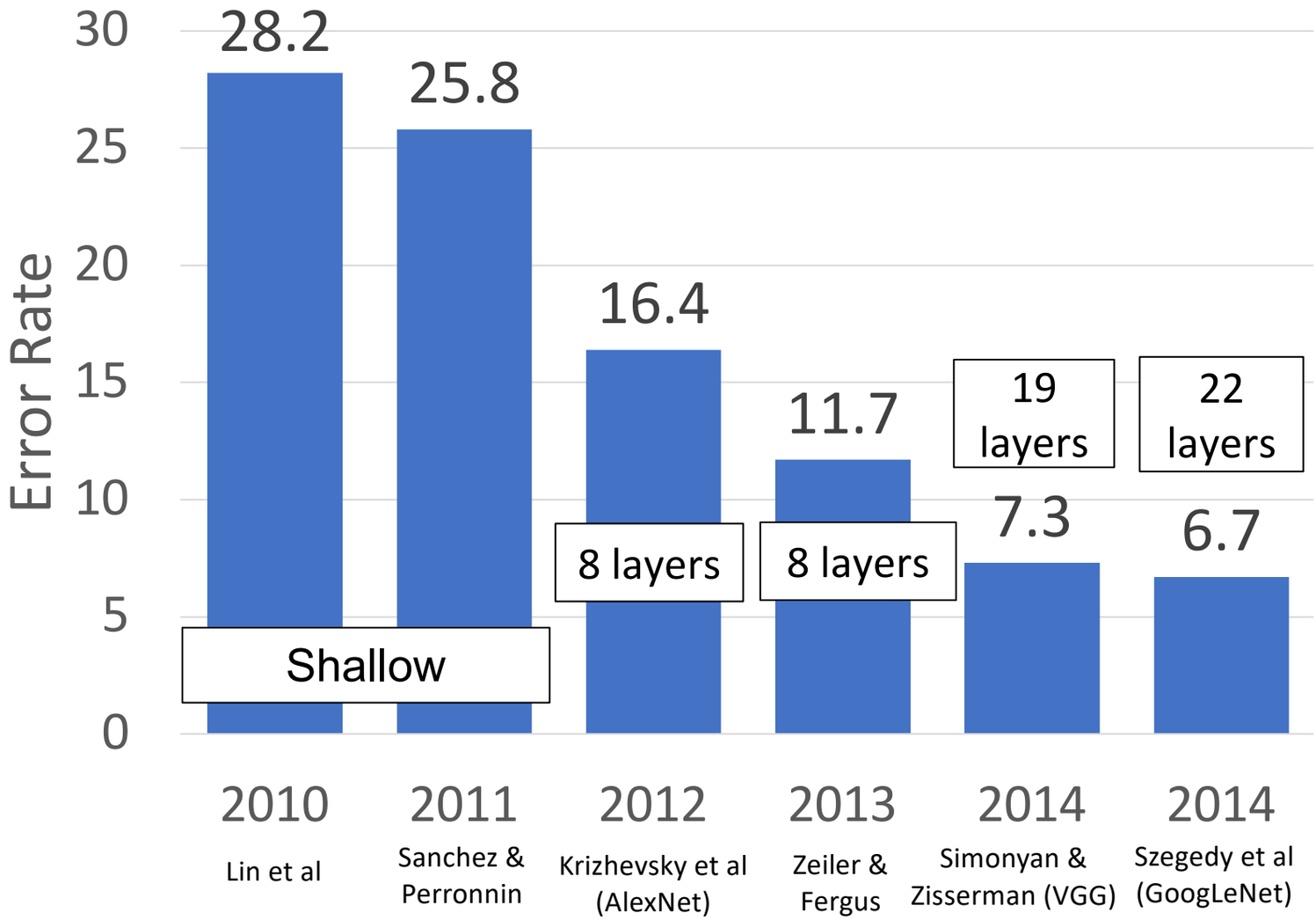
Training using loss at the end of the network didn't work well:
Network is too deep, gradients don't propagate cleanly

As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss

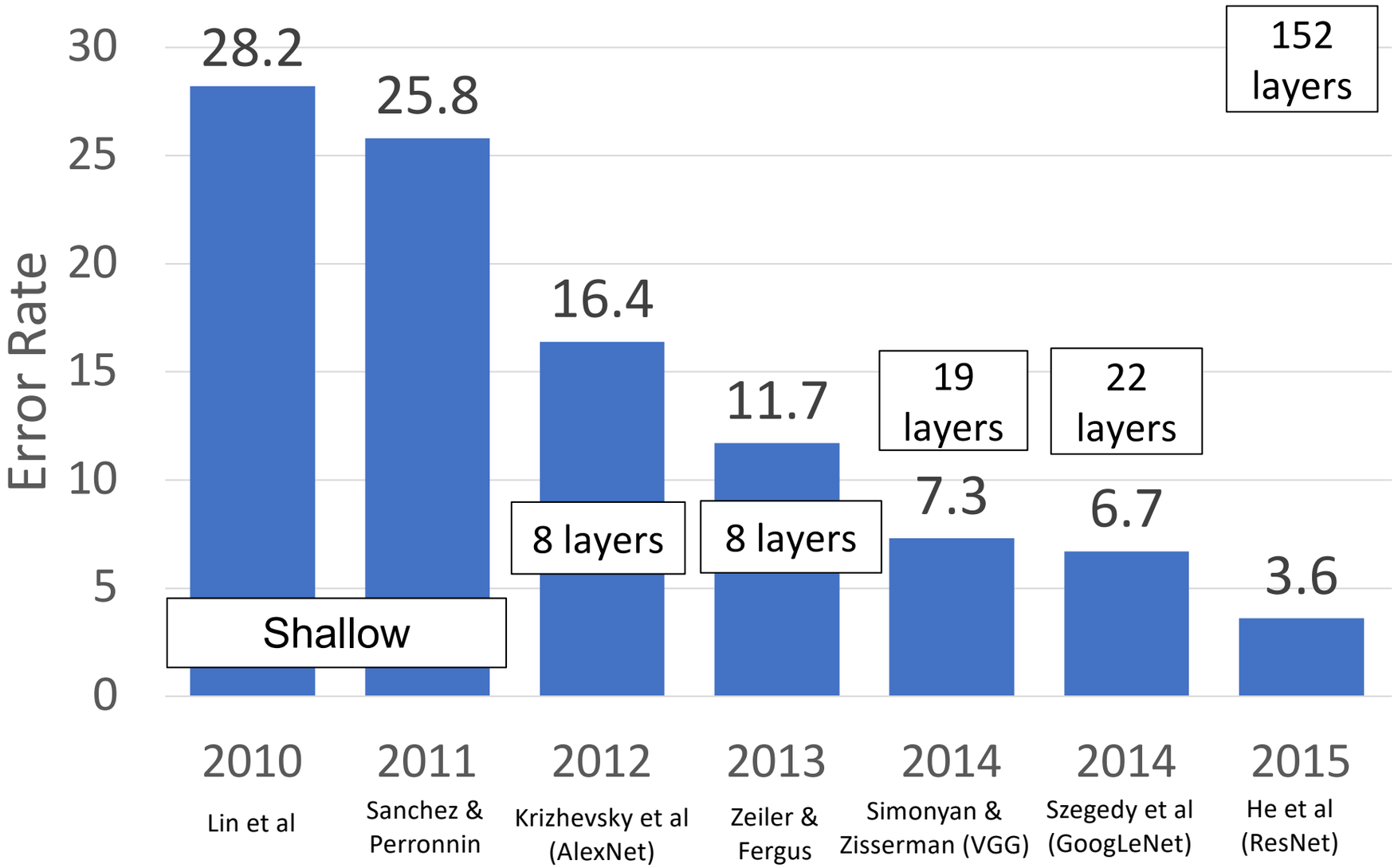
GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick



The ImageNet Challenge



The ImageNet Challenge



Residual Block

A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

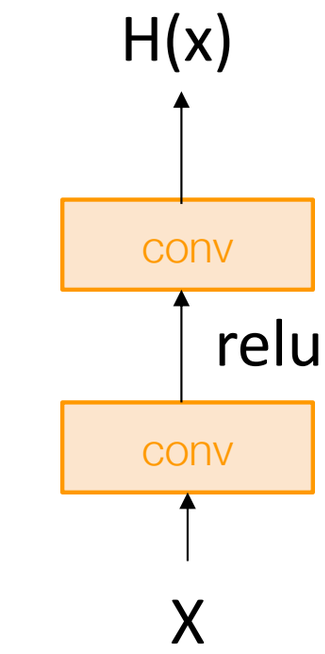
Thus deeper models should do at least as good as shallow models

Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

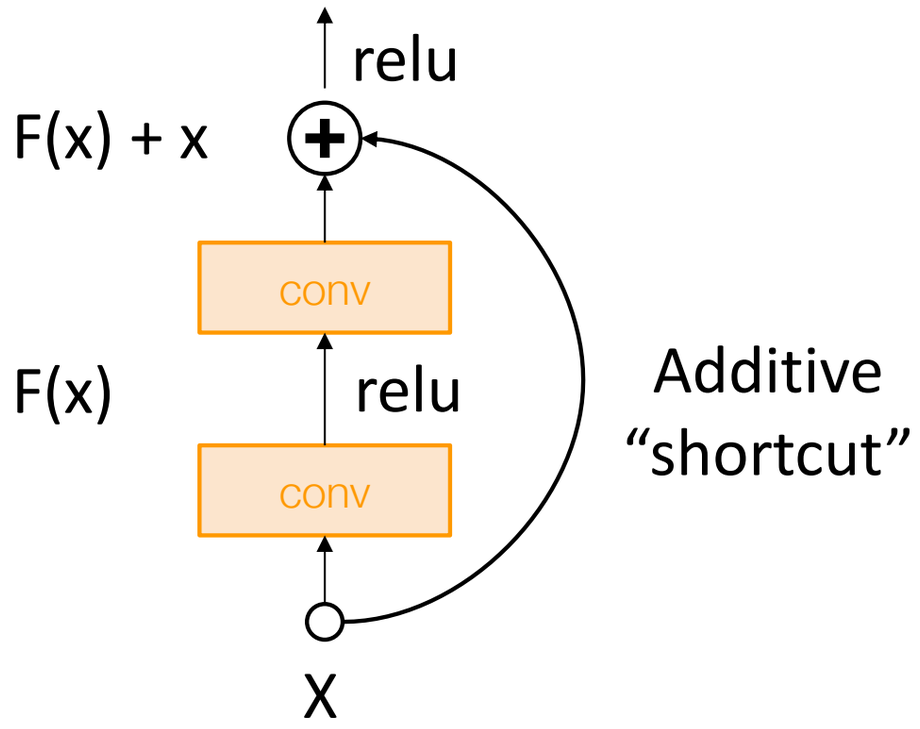
Solution: Change the network so learning identity functions with extra layers is easy!

Residual Block

Solution: Change the network so learning identity functions with extra layers is easy!



"Plain" block



Residual Block

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

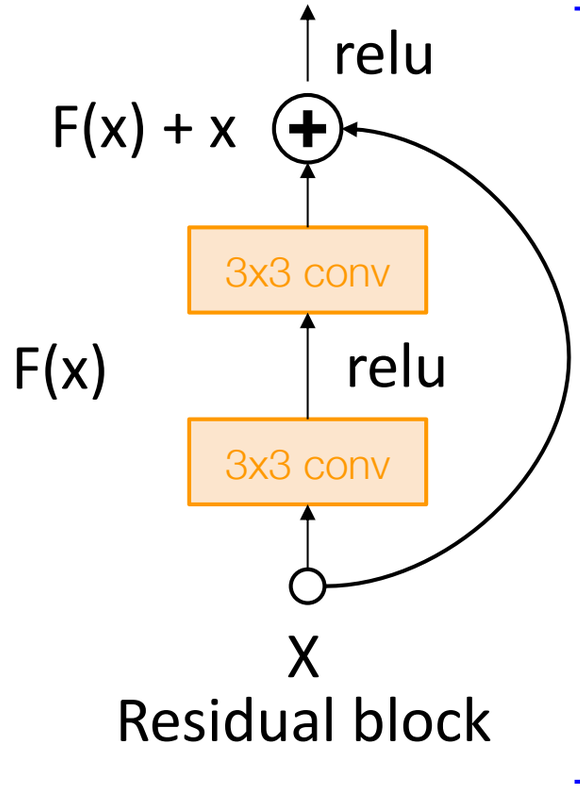
ResNet

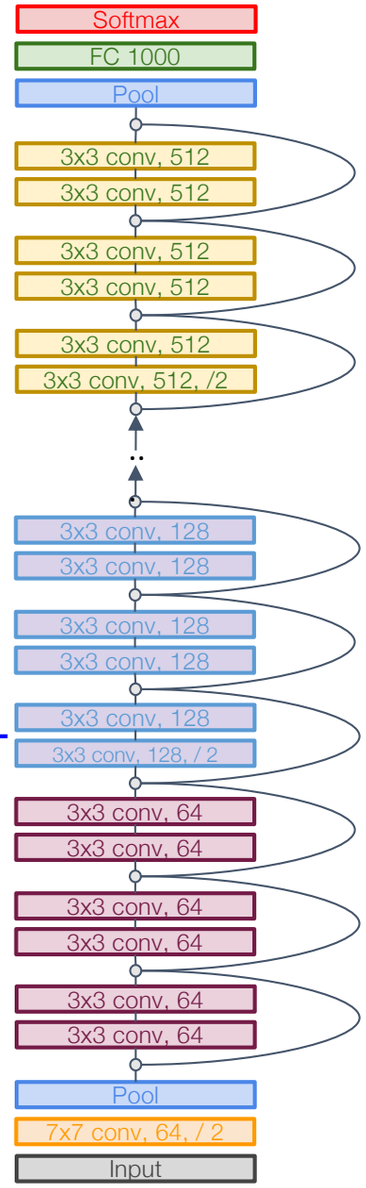
Residual Networks

A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels





He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

For vision tasks, we seem to have a good enough backbone model.
Better training methods to go deeper?

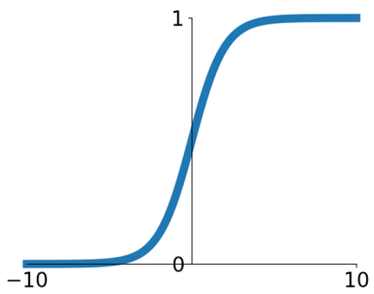
Machine Learning: IV

- Convolutional network
- Deep learning
  - Gradient vanishing
  - Deep NN architectures
  - Tricks of the trade
- Take-home messages

Activation Functions

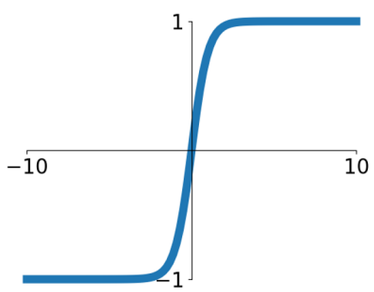
Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



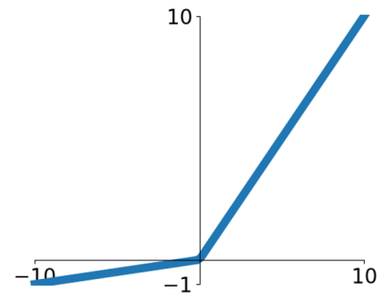
tanh

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



Leaky ReLU

$$\max(0.2x, x)$$

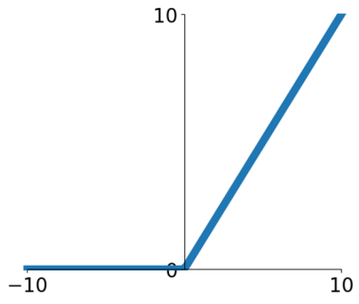


Softplus

$$\log(1 + \exp(x))$$

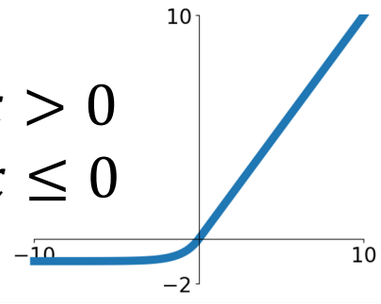
ReLU

$$\max(0, x)$$



ELU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$$



ReLU is the most common choice in today's NN learning,
The biggest advantage is to reduce gradient vanishing.

Batch Normalization

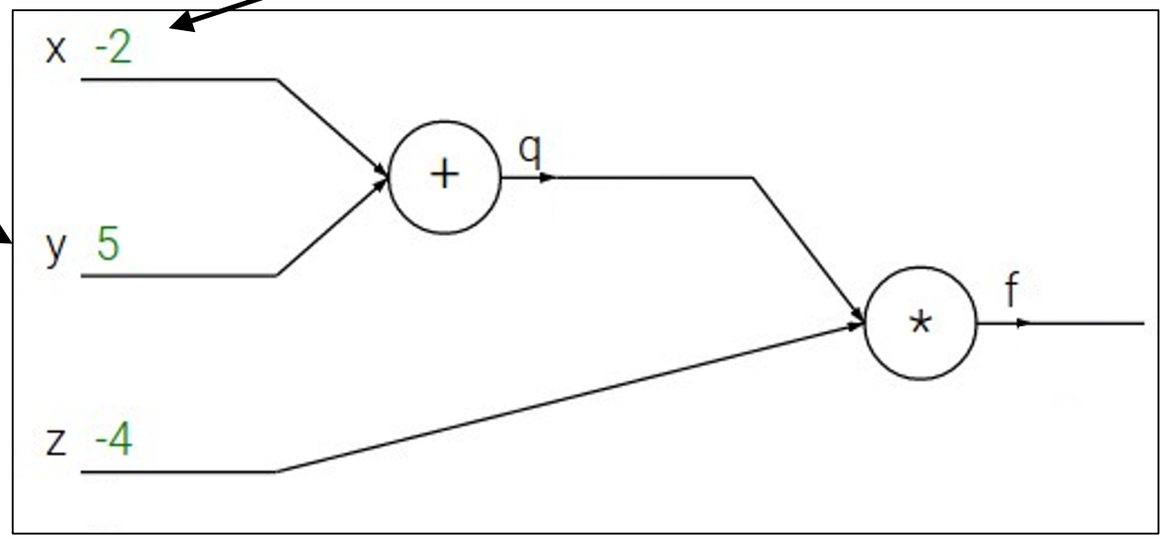
The gradient bottom layers need compound multiplication, leading to gradient vanishing (or explosion).

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$



Batch Normalization

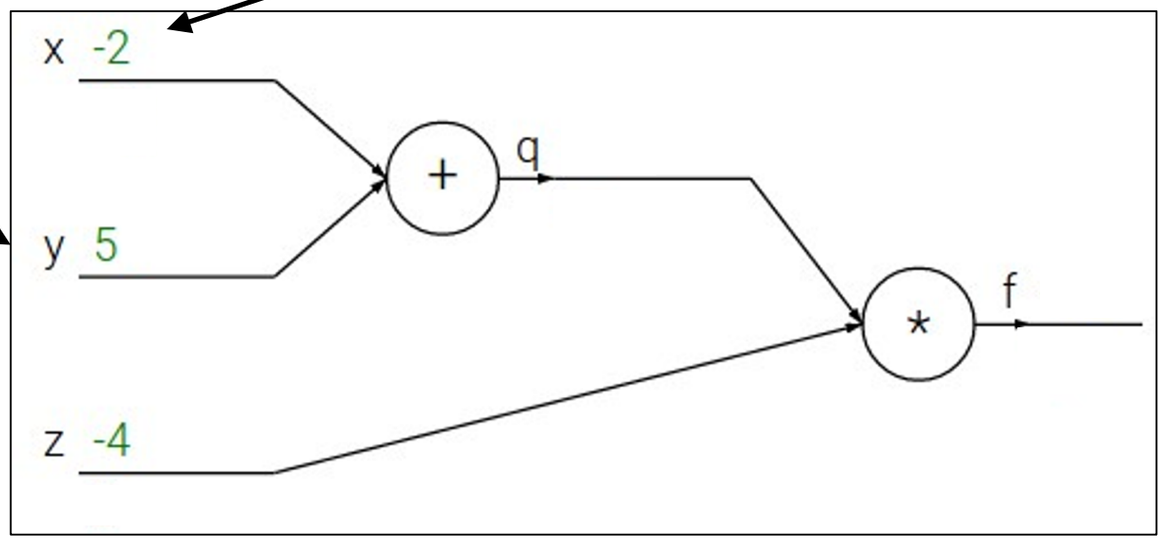
The gradient bottom layers need compound multiplication, leading to gradient vanishing (or explosion).

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$



We can normalize a batch of activations like this:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

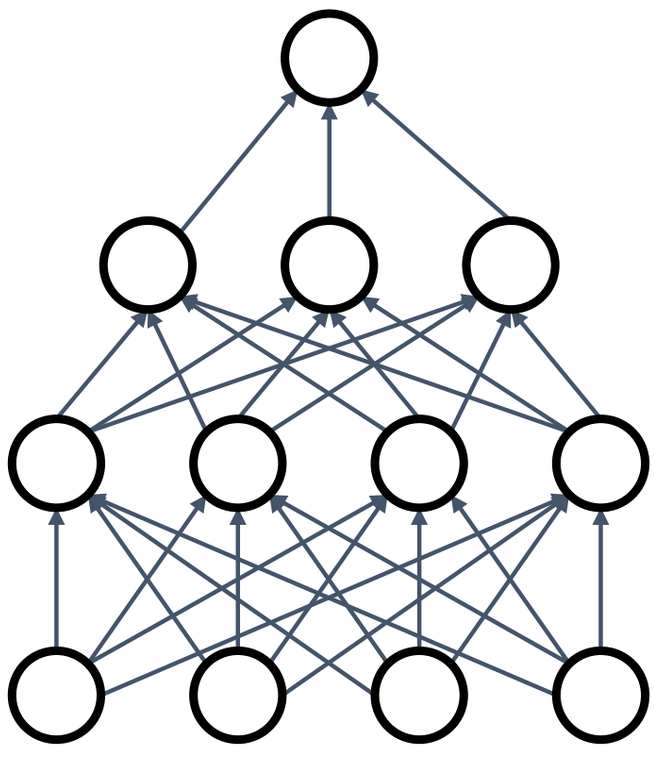
This is a **differentiable function**, so we can use it as an operator in our networks and backprop through it!

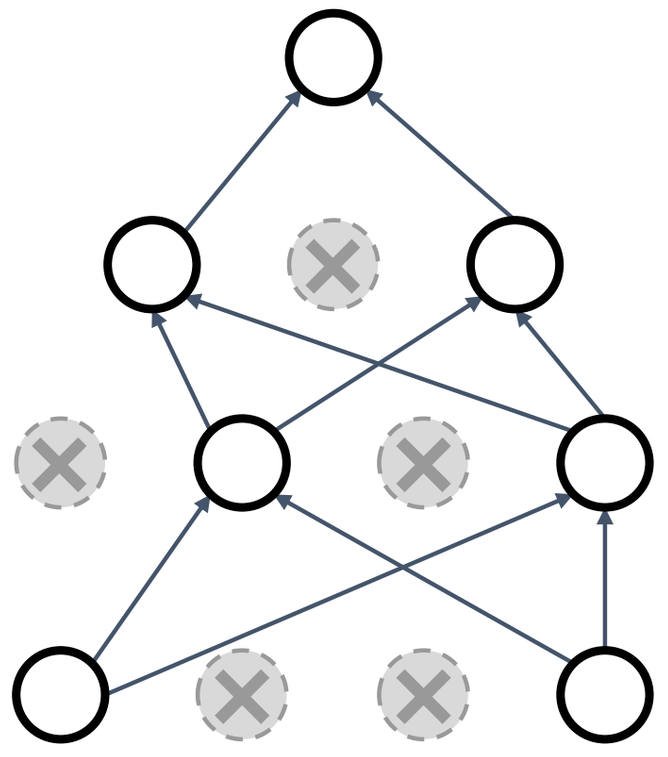
Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Dropout

Regularization: Dropout

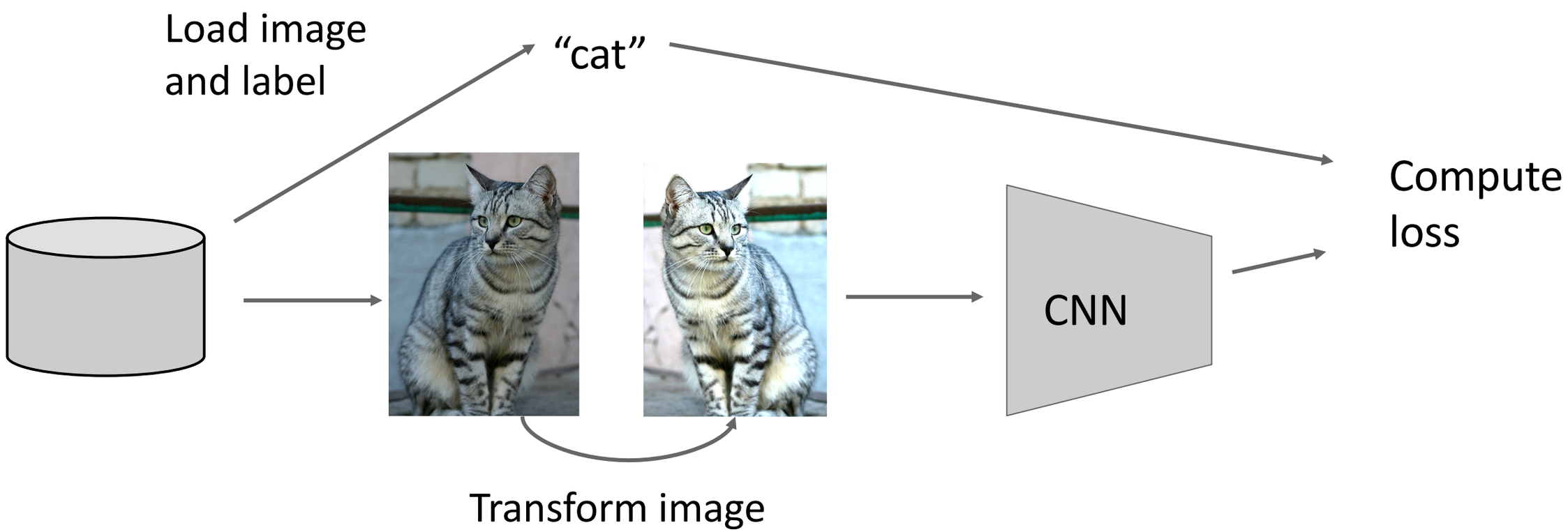
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common





Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Data Augmentation



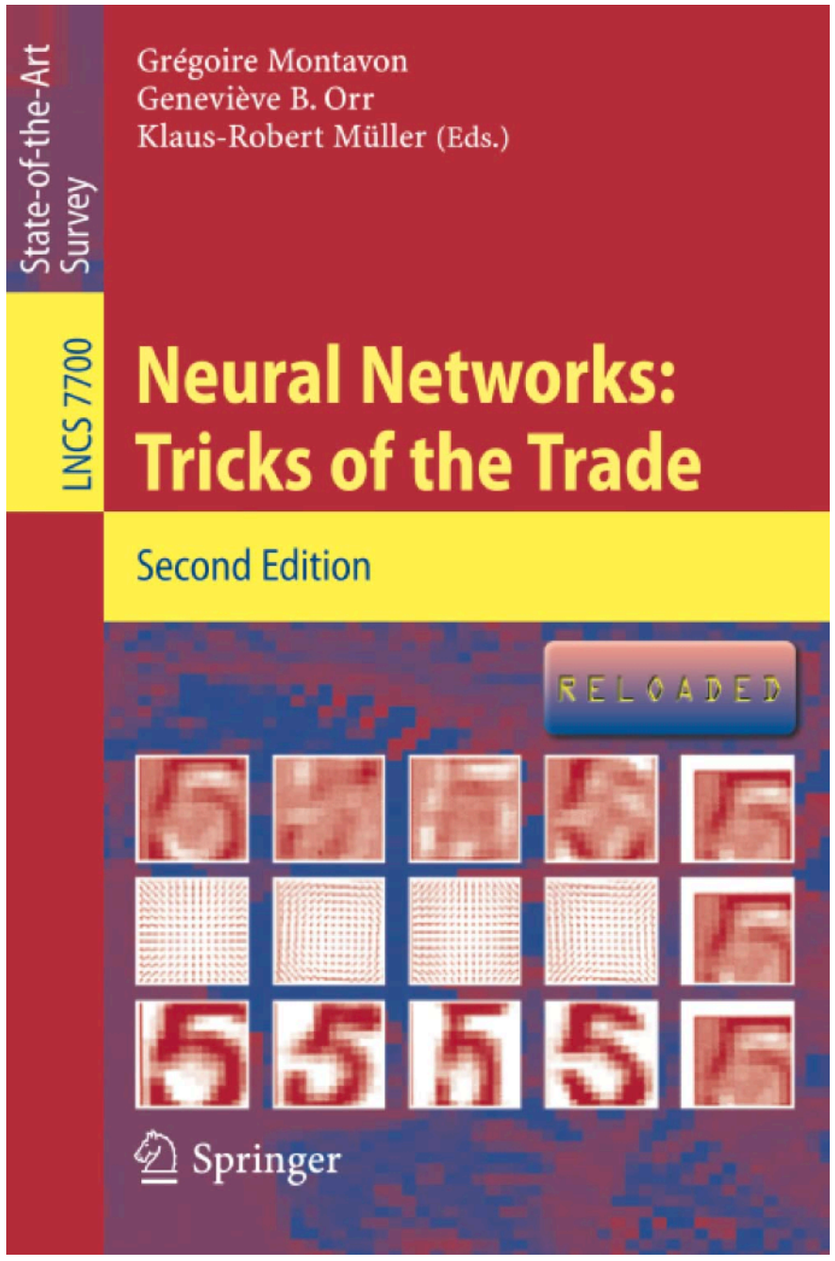
During training, randomly transform the training images (reflection, crop, reshape…)
For better generalization under style change.

Optimization Methods

Algorithm	Tracks first moments (Momentum)	Tracks second moments (Adaptive learning rates)	Leaky second moments	Bias correction for moment estimates
SGD	✗	✗	✗	✗
SGD+Momentum	✓	✗	✗	✗
Nesterov	✓	✗	✗	✗
AdaGrad	✗	✓	✗	✗
RMSProp	✗	✓	✓	✗
Adam	✓	✓	✓	✓

Variants of SGD.
The choice of optimization method is problem-dependent.

More Tricks to Work Better



Machine Learning: IV

- Convolutional network
- Deep learning
  - Gradient vanishing
  - Deep NN architectures
  - Tricks of the trade
- Take-home messages

Take-Home Messages

- Convolutional networks are built upon convolutional layers for reducing model complexity and modeling local structures of data.

- The biggest challenge of increasing the depth of NN is gradient vanishing.

- Dedicated model design and learning strategies can alleviate the gradient vanishing problem.

- After ten-years of development, deep learning has grown maturely. We will introduce some interesting models and applications in the next lecture.

Recommended Book

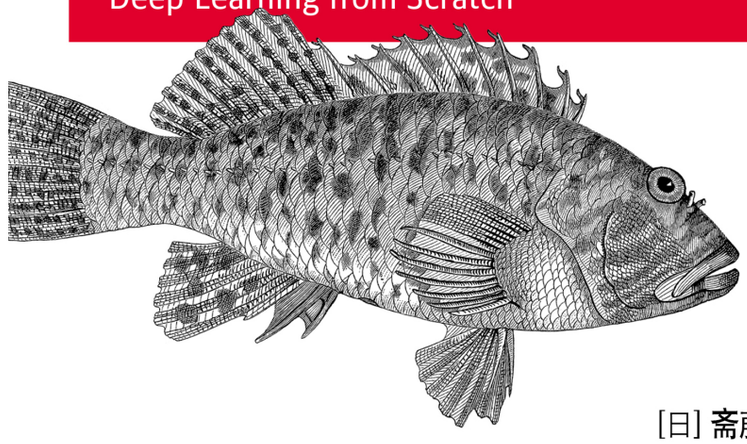
O'REILLY®
オライリー・ジャパン

TURING 图灵程序设计丛书

深度学习入门

基于Python的理论与实现

Deep Learning from Scratch



[日] 斋藤康毅 著
陆宇杰 译

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

Thanks for your attention!
Discussions?

Acknowledgement: Many materials in this lecture are taken from Justin Johnson: https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html